Janne Kari

Langattoman peliohjaimen varusohjelmisto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

9.9.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Janne Kari Langattoman peliohjaimen varusohjelmisto 52 sivua + 1 liite 9.9.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Sulautetut järjestelmät
Ohjaaja	Lehtori Keijo Länsikunnas

Insinöörityön tarkoituksena oli ohjelmoida varusohjelmisto asiakkaalta saatuun sulautettuun järjestelmään. Järjestelmästä muodostettiin peliohjain erillisenä projektina toteutettuun liikuntapeliin. Työn tavoitteena oli luoda toimiva prototyyppi annetussa aikataulussa, minkä lisäksi peliohjainta oli määrä pystyä käyttämään itsenäisesti tasapainokeppinä kuntoilun apuna. Ajatuksena oli, että peliohjain voitiin viedä esimerkiksi kuntosaleille.

Peliohjaimen runko rakennettiin mikroprosessorista ja kiihtyvyysanturista, joita täydennettiin 7-segmenttinäytöillä, värinämoottoreilla ja Bluetooth-moduulilla.

Peliohjaimesta oli olemassa kaksi prototyyppiä. Ensimmäisessä prototyypissä käytettiin piirikortteja ja moduuleita, joihin tarvitut komponentit oli integroitu. Prototyyppi ohjelmoitiin käyttämällä Arduino-ohjelmointiympäristöä ja sen valmiita kirjastoja. Toisessa prototyypissä komponentit ladottiin suoraan erilliselle piirilevylle. Lopullinen, toinen prototyyppi ohjelmoitiin puhtaasti C-ohjelmointikielellä.

Prototyyppien ohjelmointi aloitettiin kiihtyvyysanturista ja näytöistä, jotka yhdessä värinämoottoreiden kanssa muodostivat ensimmäisen testeihin menneen prototyypin. Luodun ohjelmiston kääntäminen toiseen prototyyppiin ei sujunut vaivatta, sillä valmiiden kirjastojen käyttämisen sijaan prototyyppi ohjelmoitiin alusta alkaen uusiksi. Lisäksi ongelmia ilmeni laitetasolla. Ongelmat saatiin kuitenkin ratkaistua yhteistyössä kolmannen osapuolen yrityksen kanssa, joka myös toimitti projektissa käytetyt piirilevyt.

Toiseen prototyyppiin lisättiin Bluetooth-toiminnot, jotka yhdessä ensimmäisessä prototyypissä toteutettujen toimintojen kanssa muodostivat toimivan kokonaisuuden. Peliohjaimen toimivuus testattiin niin pelin kanssa kuin ilmankin sitä, minkä lisäksi peliohjaimen asetuksia voitiin muuttaa Bluetooth-moduulin kautta ilman, että peliohjaimen lähdekoodia muokataan.

Prototyyppi luovutettiin asiakkaalle jatkokehitettäväksi keväällä 2016. Prototyyppiä oli jo mahdollista käyttää sellaisenaan, mutta asiakkaalle annettiin kehitysehdotuksia muun muassa laitepuolelle, ja prototyyppiä jatkokehitetään syksyllä 2016.

Avainsanat	mikroprosessori, kiihtyvyysanturi, peliohjain, sulautettu ohjelmointi

Author Title Number of Pages Date	Janne Kari Firmware of a wireless game controller 52 pages + 1 appendix 9 September 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Systems
Instructor	Keijo Länsikunnas, Senior Lecturer

This thesis focuses on the creation of firmware for an embedded system used as a game controller in a sports game. In a given schedule, the goal of the project was to create a working prototype which is not dependent on the game. This allowed the controller to be used as a balance stick without the game, for example at gyms.

The core of the controller was built from a microprocessor and an accelerometer. The controller also contained 7-segment displays, vibration motors and a Bluetooth module. There were two prototypes of the controller. The first prototype was programmed using pre-built libraries included in the Arduino Software while the second one was done using the C programming language.

The creation of the firmware started from the accelerometer and the displays which together with the vibration motors formed the first prototype. In addition to the features included in the first prototype, the second prototype also had Bluetooth features. The controller was tested thoroughly with and without the game and the configurations of the prototype could be changed through the Bluetooth without altering the source code.

The project itself was a huge learning process and the final prototype was handed over to the customer in April 2016. The customer was also presented with ideas about improving the hardware. The prototype will be developed further in the autumn of 2016.

Keywords	microprocessor, accelerometer, game controller, embedded programming

Sisällys

Lyhenteet

BLE	Bluetooth Low Energy
BOD	Brown-out Detection
EEPROM	Electronically Erasable Programmable Read-Only Memory
GND	Ground
I^2C	Inter-Integrated Circuit
ISP	In-System Programming
MISO	Master In, Slave Out
MOSI	Master Out, Slave In
SCL, SCK	Serial Clock Line
SDA	Serial Data Line
SS	Slave Select
TWCR	TWI Control Register
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver/Transmitter
V_{CC}	Voltage at the common collector, positiivinen jännitelähde
WDT	Watchdog Timer

1 Johdanto

Insinöörityönä tehdään varusohjelmisto sulautettuun järjestelmään, joka muodostaa peliohjaimen erillisenä projektina toteutettavaan peliin. Peliohjain, joka on muodoltaan tasapainokeppi, sisältää mikroprosessorin ja kiihtyvyysanturin, jotka muodostavat kepin rungon. Keppi sisältää myös 7-segmenttinäytöt ja värinämoottorit, joita käytetään interaktiivisuuteen kepin ja käyttäjän välillä. Tiedonsiirto toteutetaan Bluetooth-moduulin avulla.

Insinöörityön tavoitteena on luoda annettuun päivämäärään mennessä prototyyppi, jota on mahdollista käyttää sekä pelin kanssa että ilman kyseistä peliä. Tarkoituksena on, että keppiä voidaan käyttää pelin pelaamisen ohella kuntoilun apuna esimerkiksi kotona ja kuntosaleilla. Siten toteutuksessa joudutaan luultavasti tekemään kompromisseja, jotta keppi toimii molemmissa käyttötarkoituksissa.

Peliohjainta tehdään Metropolia Games -pelistudion toimesta DeRing Oy:lle. Projekti aloitetaan 8.2.2016 ja se päättyy viimeistään 28.4.2016. Keppiin saadaan valmis laitteisto kolmannen osapuolen yritykseltä, joka on samalla vastuussa tämän projektin laitteistopuolesta. Ohjaimeen tulleeseen laitteistoon ei voitu vaikuttaa ja keppiin ohjelmoitava rajapinta, jonka tekeminen on tämän projektin pääasiallinen työtehtävä, toteutetaan sille määräytyneiden laitteistorajoitteiden mukaisesti.

Lupa tämän insinöörityön tekemiseen on saatu DeRing Oy:ltä. Projektissa tehtävä prototyyppi muuttuu vielä ennen sen julkaisua, joten insinöörityössä esiteltyä prototyyppiä ja tuotantoon menevää, markkinoille julkaistavaa tuotetta ei voi suoraan verrata keskenään. Tasapainokeppi tulee markkinoille vasta myöhemmin, joten keppiin tulevaa koodia ei julkaista tämän insinöörityöraportin yhteydessä.

2 Laitteiston toiminta

Tasapainokeppi valmistettiin insinöörityönä peliin, joka tuotetaan Metropolia Games -pelistudion kautta erillisenä projektina. Pelin tarkoituksena on seurata pelissä näkyvää hahmoa, joka esiintyy erilaisille yleisöille erilaisissa maisemissa. Hahmo tekee vaihtelevia liikesarjoja, joita pelaajan on tarkoitus matkia toistamalla liikkeet sellaisenaan. Tarkoituksena on, että pelaaja pitää tasapainokepin, jota pelissä käytetään ohjaimena, tasapainossa liikkeiden aikana. Tasapainon tarkkailemiseksi laitteiston suunnittelija laittoi keppiin kiihtyvyysanturin, jonka avulla kallistuksen mittaaminen on mahdollista.

Tasapainokepistä tehtiin kaksi prototyyppiä. Prototyypeissä käytetyt komponentit ovat pohjimmiltaan samoja, mutta ensimmäisessä prototyypissä käytettiin erilaisia moduuleita, joihin esimerkiksi kiihtyvyysanturi ja Bluetooth-moduuli oli integroitu. Näin ollen prototyyppien ulkonäkö eroaa täysin, vaikka pohjimmiltaan molemmissa käytettiin samaa laitteistoa. Prototyypeissä käytettiin myös erilaisia kehittämistapoja. Ensimmäinen laite ohjelmoitiin käyttämällä Arduinon ohjelmointiympäristöä, kun taas toinen ohjelmointiin puhtaasti C-kielellä käyttämällä Atmel Studio 6.2:ta.

Sen lisäksi, että tasapainokeppiä käytetään hyväksi erillisessä pelissä, on keppiä mahdollista käyttää myös itsenäisesti ilman muuta laitteistoa. Projektin tarkoituksena olikin, että keppiä olisi mahdollista käyttää kuntoilun apuna kotona tai kuntosaleilla, esimerkiksi osana lämmittelyä tai erilaisia liikeratoja ja liikkuvuusharjoitteita. Yleiskäyttöisyyden takia projektissa jouduttiin tekemään muutamia kompromisseja, jotta tuote saataisiin toimivaksi niin pelissä kuin ilmakin sitä.

2.1 Kiihtyvyysanturi

Projektissa käytettiin kiihtyvyysanturina Freescale Semiconductorin valmistamaa kolmiakselista MMA8451QR1-kiihtyvyysanturia. Käytetty kiihtyvyysanturi on 14-bittinen eli sen numerolliseksi tarkkuudeksi voidaan määritellä $2^{14} = 16384$ [1, s. 1].

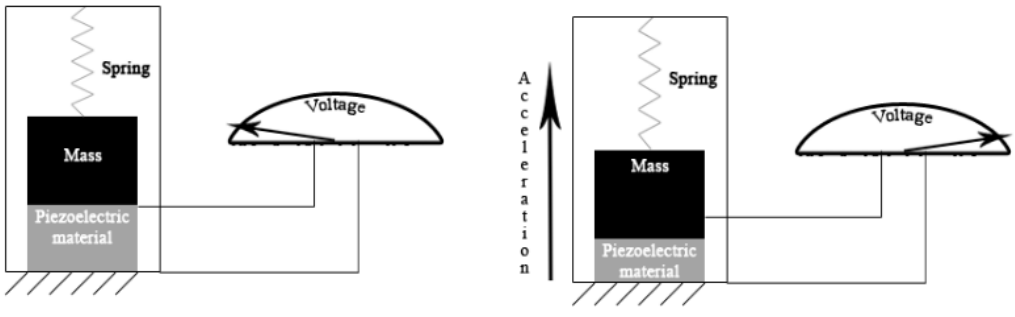
Kiihtyvyysanturin tarkoitus on nimensä mukaisesti mitata kappaleen kiihtyvyyttä eli kappaleen nopeuden muutosta suhteessa aikaan. Kiihtyvyyden yksikkö on m/s^2, ja sen yhteydessä puhutaan yleensä niin sanotuista G-voimista. G-voimilla tarkoitetaan kiihty-

vyyttä suhteessa maan painovoimaan, ja 1 G on yleisesti suuruudeltaan 9,81 m/s^2, joskin sen suuruus vaihtelee esimerkiksi eri puolilla maapalloa. [2.]

Anturia on mahdollista käyttää ±2g-, ±4g- ja ±8g-tarkkuudella, joka tarkoittaa sitä, että 2^{14} bitin osuus on jaettu erilaisiin osiin. ±2g-tarkkuus on jaettu neljään osaan, mikä tarkoittaa, että jokaista G:tä kohden on tarkkuutta 16384 / 4 = 4096, ±4g-tarkkuudella 16384 / 8 = 2048 ja ±8g tarkkuudella 16384 / 16 = 1024. [1, s. 10.] Tarkkuuden valinta tulee tehdä sen mukaan, missä kiihtyvyysanturia on määrä käyttää. Jos kiihtyvyysanturiin ei tule kohdistumaan suuria voimia, on järkevintä käyttää ±2g-tarkkuutta. Jos taas kiihtyvyysanturiin tulee kohdistumaan voimia, jotka ylittävät ±2g- tai ±4g-voimat, on anturi syytä asettaa toimimaan ±8g-tarkkuudella. [2.]

Kiihtyvyysantureiden toiminta perustuu yleensä kapasitiivisiin levyihin, joista osa on kiinnitetty kiinteästi paikalleen ja osa on kiinnitetty jousiin, jotka liikkuvat niihin kohdistuvien voimien ansiosta. Kun etäisyys kapasitiivisten levyjen välillä muuttuu, muuttuu samalla niiden välinen kapasitanssi, josta on mahdollista määrittää kiihtyvyysvektorin suunta ja koko. [3.] Näin on toteutettu myös MMA8451QR1-kiihtyvyysanturi, mikä on mahdollista huomata datalehden etusivulla olevasta maininnasta ja datalehden sivulla 3 olevasta ”C to V Converter” -muuntimesta (Coulomb to Voltage Converter, sähkövaraus jännitteeksi).

Toinen mahdollinen tapa on käyttää kuvan 1 mukaisesti pietsosähköisiä materiaaleja.



Kuva 1. Pietsosähköisen materiaalin toiminta kiihtyvyysanturissa [2].

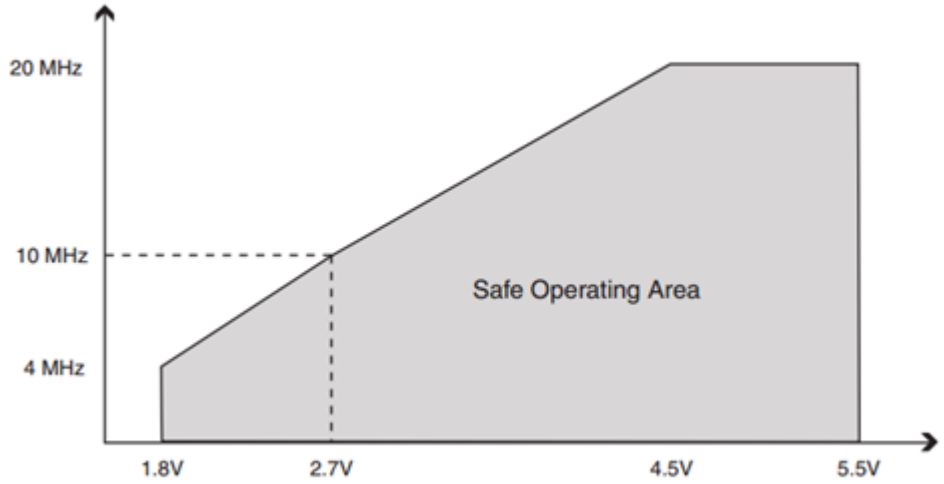
Pietsosähköiset materiaalit tuottavat elektronisen varauksen, kun ne altistetaan mekaaniselle rasitukselle.

2.2 Prosessori

Projektissa oli käytössä Atmelin vuonna 2009 julkaisema 8-bittinen ATmega328P-mikroprosessori [4, s. 650]. Prosessorin flash-muistin suuruus on 32 kilotavua, kun taas EEPROM (Electronically Erasable Programmable Read-Only Memory) oli kooltaan yhden kilotavun ja SRAM (Static Random Access Memory) kaksi kilotavua [5]. Näistä EEPROM-muistille kirjoitetaan prosessoriin liittyvät asetustiedot, kun taas flash-muistille kirjoitetaan ohjelmakoodit ja SRAM-muistille muuttujat, joita ohjelmakoodi käyttää ohjelman toteutuksessa [6].

Prosessoria on mahdollista käyttää erilaisilla käyttöjännitteillä, aina 1,8 V:sta 5,5 V:iin. Prosessori sisältää sisäisen kellokiteen, joka yltää enimmillään 8 MHz:n taajuuteen, mutta prosessoria on mahdollista käyttää myös ulkoisella oskillaattorilla. Tällöin kellotaajuus rajoittuu 20 MHz:iin, joka on määriteltynä raja-arvoksi Atmelin dokumentoinnissa. [4, s. 28; 5.]

Käytetyn jännitteen ja sitä vastaavan turvallisen kellotaajuuden käyttö on esiteltynä kuvassa 2. Liian alhaisen käyttöjännitteen ja liian korkean kellotaajuuden yhdistäminen saattaa aiheuttaa ongelmia prosessorille muun muassa epävakaana toimintana ja tulosten vääristymisenä. Lisäksi se voi aiheuttaa EEPROM-muistin korruptoitumisen, joka johtaa asetustietojen vääristymiseen ja pahimmassa tapauksessa prosessorin käyttökelvottomuuteen [4, s. 21].



Kuva 2. Prosessorin käyttöjännitteen ja turvallisen kellotaajuuden yhteys [4, s. 303].

Prosessorissa on yhteensä 32 nastaa, joista 23 on yleiskäyttöisiä I/O-nastoja (Input/Output). I/O-nastoja voidaan käyttää datan kuljettamisessa molempiin suuntiin. Loput yhdeksän nastaa on tarkoitettu muun muassa käyttöjännitteille, maapisteille ja kellosignaalille. [4, s. 3–5.] Lisäksi prosessori sisältää 26 erilaista keskeytysvektoria [4, s. 65]. Keskeytysvektorilla tarkoitetaan tietynlaista lippua siitä, että jokin asia on tapahtunut. Prosessori pitää kirjaa lippujen tiloista, ja lipun noustessa ylös prosessori tietää siirtyä kyseisen keskeytysvektorin määrittelemään funktioon. [4, s. 14–15.]

Sulakkeet

ATmega328P-prosessorissa on käytössä yhteensä 11 erilaista sulaketta (fuse), joilla prosessorin toimintaa on mahdollista säätää laitetasolla. Sulakkeiden avulla prosessori saadaan toimimaan omaan käyttötarkoitukseen sopivalla tavalla, joten niiden käyttö oli oleellista projektin kannalta. [7.]

Liitteessä 1 on esitelty lista kaikista sulakkeista, joita Atmelin prosessoreissa on käytössä. Projektin kannalta tärkeimmät sulakkeet tulevat luultavasti olemaan BODLEVEL, CKDIV8 ja SUT_CKSEL. BODLEVEL-sulakkeen (Brown-out Detection) tarkoituksena on pitää kirjaa prosessoriin menevästä syöttöjännitteestä. Jos jännitetaso alittaa sulakkeessa määritellyn tason, sammuttaa sulake prosessorin välittömästi. Näin saadaan

vältettyä saadun datan vääristyminen ja prosessorin muistissa olevien asetustietojen korruptoituminen [4, s. 21].

ATmega328P-prosessorissa on sisäinen oskillaattori, joka tuottaa enimmillään 8 MHz:n kellotaajuuden [4, s. 28]. CKDIV8-sulakkeen tarkoituksena on jakaa kellotaajuus kahdeksanteen osaan, eli 8 MHz:n taajuudesta saadaan aikaiseksi 1 MHz:n taajuus [7; 8]. Sulake on asetettu tehdasasetuksissa päälle, jotta prosessori toimisi ongelmitta suoraan tehtaalta tuotuna jokaisella sallitulla käyttöjännitteellä. Prosessorin sallitut käyttöjännitteet ovat väliltä 1,8–5,5 V, joten alimmalla mahdollisella käyttöjännitteellä ei ole turvallista käyttää täyttä 8 MHz:n kellotaajuutta, kuten kuvasta 2 voi nähdä.

SUT_CKSEL-sulake määrittelee prosessorin käynnistymisajan ja mahdollistaa kellotaajuuden valitsemisen sisäisistä ja ulkoisista oskillaattoreista [7; 8]. Lopulliseen prototyyppiin on määrä tulla ulkoinen 16 MHz:n oskillaattori, joten tarkoituksena on poistaa CKDIV8-sulake käytöstä ja vaihtaa prosessori käyttämään ulkoista oskillaattoria asettamalla SUT_CKSEL-sulake tilaan EXTXOSC. Tämä saattaa kuitenkin aiheuttaa ongelmia. Prototyypissä on käytössä 3,3 V:n käyttöjännite, joka ei Atmelin dokumentaation mukaan riitä turvalliseen 16 MHz:n käyttöön [kuva 2]. Tämä tullaan kuitenkin varmistamaan käytännössä, minkä jälkeen selviää, onko 16 MHz:n taajuutta mahdollista käyttää luotettavasti lopullisessa tuotteessa vai tuleeko käytettyä kellotaajuutta pienentää.

Mahdolliset ongelmat

ATmega328P-prosessorin sisältämän kahden kilotavun SRAM-muistin määrä ei välttämättä riitä. Tasapainokepin paikallisiin toimiin muistimäärä riittää teoriassa hyvin, mutta BLE-moduulin (Bluetooth Low Energy) ja prosessorin välistä rajapintaa ei ole standardoitu. Tämä saattaa aiheuttaa ongelmia muistimäärän suhteen, sillä etukäteen ei voida määritellä, kuinka paljon moduulin käyttämä rajapinta vaatii muistia.

ATmega328P-prosessori tulee tehtaalta asetuksella, jossa prosessorin kellotaajuus on 1 MHz. Tämä on toteutettu asettamalla päälle CKDIV8-sulake, joka jakaa sisäisen kellotaajuuden kahdeksalla. Prosessori ei myöskään osaa automaattisesti käyttää ulkoista oskillaattoria, joten prototyypissä oleva oskillaattori ei ole käytössä, ennen kuin sulakkeita käydään muuttamassa. [7; 8.] Tämä saattaa aiheuttaa ongelmia sarjatuotannos-

sa, sillä pelkkä ohjelman lataaminen prosessoriin ei tässä tapauksessa riitä vaan sulakkeiden tilat on asetettava prototyypeissä käytettyjen tilojen mukaisiksi.

Kolmas mahdollisesti ongelmia aiheuttava seikka on prototyypissä olevan ulkoisen oskillaattorin kellotaajuus. Prosessoria tullaan käyttämään 3,3 V:n käyttöjännitteellä, mutta kuten kuvasta 2 on mahdollista nähdä, on käyttöjännite liian pieni 16 MHz:n kellotaajuuden turvalliseen käyttöön. Oskillaattorin toiminta kokeillaan, mutta pahimmassa tapauksessa prototyypissä oleva prosessori rikkoutuu kokeilun tuloksena.

2.3 Bluetooth-moduuli

Projektissa käytetty Bluetooth-moduuli oli Laird Technologiesin BL600-SA-moduuli. Moduuli oli toteutukseltaan BLE-tyyppinen (Bluetooth Low Energy), ja sen tarkoituksena on tuottaa samanlaiset ominaisuudet kuin tavallisessa Bluetoothissa, kuitenkin pienemmällä käyttöteholla. BLE-moduuli on hyvä valinta käytettäväksi laitteessa, jossa käytetään kannettavaa virtalähdettä, tässä tapauksessa akkua. [9.] Moduuli tuottaa kuitenkin samalla joitakin haasteita.

Moduuli toimii 2,4 GHz:n radiotaajuudella, joka vastaa tavallisen Bluetoothin käyttämää taajuutta [10]. Samalla radiotaajuudella toimivat muun muassa WLAN-verkot, joten alueilla, joilla kyseinen taajuus on laajassa käytössä, saattaa aiheutua yhteysongelmia ruuhkautumisen takia [11]. Teoreettiseksi tiedonsiirtonopeudeksi BL600-SA-moduulille on määritelty ilmassa 1 megatavu sekuntia kohden [10].

Moduulin sijainti tulee luultavasti vaikuttamaan sen tehonkulutukseen ja kantavuuteen, sillä tiedonsiirtoon käytetty antenni on sijoitettu moduulin päälle. Vähäinen virta tarkoittaa rajoitetumpaa lähetystehoa [12]. Tässä tapauksessa moduuli suljetaan tiiviin alumiinirakenteen sisälle. Näin moduulin antenni jää rakenteiden sisälle eikä antennilla ole suoraa näköyhteyttä kohdelaitteeseen. Laird Technologies ei anna moduulille minkäänlaista luvattua kantavuutta, mutta yleisesti ottaen BLE-moduuleiden on luvattu kantavan yli 30 metriin [9; 13]. Tämä on kuitenkin käytännössä harvinaista, sillä luvattuihin etäisyyksiin on päästy lähinnä esteettömissä laboratorio-olosuhteissa. Siten moduulin todellista kantamaa ei voida tietää, ennen kuin moduuli on suljettu lopulliseen koteloonsa ja sen kantavuus on testattu käytännössä.

Virrankulutukseksi moduulille luvataan lepotilassa 3,5 µA ja lähetystilassa 10,5 mA silloin, kun lähetysteho on 0 dBm [10]. Mahdolliset esteet ja etäisyyden kasvaminen laitteiden välillä aiheuttavat sen, että lähetystehoa joudutaan kasvattamaan moduulin päästä, jolloin virrankulutus todennäköisesti kasvaa jonkin verran [14]. Lisäksi moduulia joudutaan käyttämään jatkuvasti sen sijaan, että se lähettäisi tietoja satunnaisesti. Tämä saattaa näkyä virrankulutuksessa, mutta todelliset lähetystehot ja virrankulutukset nähdään vasta silloin, kun tuote on valmis, eikä niitä voida suoraan määrittää ennalta.

BL600-SA-moduuli ohjelmoidaan sisäisesti käyttämällä smartBASIC-ohjelmointikieltä [10]. BLE-toteutuksissa käytettyihin rajapintoihin moduulien ja prosessorien välillä ei liity standardeja, joten moduulin ohjelmointi riippuu täysin Laird Technologiesin tekemistä valinnoista. Näin ollen moduulin ohjelmointia ei voida verrata muiden vastaavien moduuleiden ohjelmointiin ja ainoa tietolähde on tässä tapauksessa yrityksen antamat dokumentaatiot. Pahimmassa tapauksessa moduulille joudutaan rakentamaan alusta alkaen omat protokollat, joiden avulla yhteys moduulin ja pelin ohjelmointiin käytetyn Unityn välille muodostetaan. Lisäksi tiedonsiirron tulee toimia molempiin suuntiin, joten ongelma ei liity enää pelkästään moduuliin vaan yhteistyötä joudutaan tekemään myös pelin kehittäjien kanssa.

Moduuli sisältää tiedonsiirtoon erilaisia rajapintoja. GPIO-linjoja (General Purpose Input/Output) on 28, SPI-linjoja kolme (Serial Peripheral Interface), I^2C-linjoja kaksi (Inter-Integrated Circuit) ja ADC-linjoja kuusi (Analog/Digital Converter) [15]. Toisin sanoen moduulia on mahdollista käyttää prosessorin tavoin. Moduuli sisältää kuitenkin myös UART-rajapinnan (Universal Asynchronous Receiver/Transmitter), jota tullaan käyttämään tiedonsiirtoon moduulin ja ATmega328P-prosessorin välillä [16].

2.4 Rajapinta

Tasapainokepin rajapintaa ohjelmoitaessa tulee toteuttaa neljä erilaista kokonaisuutta:

- kiihtyvyysdatan siirtäminen kiihtyvyysanturilta prosessorille

- luetun datan muuntaminen kallistuskulmaksi

- halutun datan kirjoittaminen 7-segmenttinäyttöihin

- tiedonsiirto Bluetoothin välityksellä tasapainokepistä peliin.

Kiihtyvyysdatan siirtämiseen tullaan käyttämään TWI-protokollaa (Two Wire Interface), jota voidaan pitää identtisenä I^2C-protokollaan verrattuna [17]. Tiedonsiirtoa varten tarvitaan kaksi linjaa kiihtyvyysanturin ja prosessorin välille. Näistä linjoista toinen on tarkoitettu tiedonsiirtokanavaksi, kun taas toinen on varattu tiedonsiirron rytmittämiseen käytettävää kellosignaalia varten. Tiedonsiirto noudattaa half-duplex-periaatetta, eli tietoa voidaan siirtää kerrallaan vain jompaankumpaan suuntaan. [1, s. 16–17.]

Kiihtyvyysanturi on tarkkuudeltaan 14-bittinen, joten x-, y- ja z-suuntainen kiihtyvyys jaettu kahteen erilliseen 8-bittiseen rekisteriin, joista toinen sisältää kahdeksan isointa bittiä (MSB, most significant bit) ja toinen kuusi pienintä bittiä (LSB, least significant bit) [1, s. 19]. Näin ollen molemmat rekisterit tulee ensin lukea esimerkiksi erillisiin muuttujiin, joista muodostetaan 14-bittinen luku. Kiihtyvyysdatan muodostamisen jälkeen kallistuskulma tulee laskea niin, että kepin kiertoakselilla ei ole merkitystä kallistuskulman laskemisessa. Tämä on tärkeää kepin käytettävyyden kannalta. Toisin sanoen kallistuskulmaan ei saa vaikuttaa mikään muu kuin itse kallistus, joten laskennassa tulee ottaa huomioon jokainen akseli.

Tasapainokeppiin on yhdistetty kolme erillistä 7-segmenttinäyttöä, jotka on tarkoitus saada toimimaan yhdessä. Asiakkaan kanssa on sovittu, että näytöt toimivat eräänlaisena laskurina, joka kertoo tasapainokepin käyttäjälle virhepisteiden määrän. Haasteeksi muodostuukin saada kaikki kolme näyttöä toimimaan yhtenä kokonaisuutena niin, että käyttäjä ei huomaa niiden olevan erillisiä yksiköitä.

Viimeisenä asiana tulee toteuttaa tiedonsiirto tasapainokepin ja pelaamiseen käytettävän laitteen välillä. Tiedonsiirron tulee olla kaksisuuntainen, eli tietoa tulee voida välittää kepistä peliin ja pelistä keppiin. Lisäksi yhteyden muodostaminen pitää saada luotettavaksi. Tasapainokeppi on akkukäyttöinen, joten virrankulutuksen minimointi nousee suureen osaan ja Bluetooth-moduulin mainostus tulee toteuttaa niin, että se ei kuluta jatkuvasti virtaa. Samoin tiedonsiirto pitää minimoida ja siirtoa tulee toteuttaa vain silloin, kun sille on oikeasti tarvetta, jotta akkua ei kuluteta turhaan.

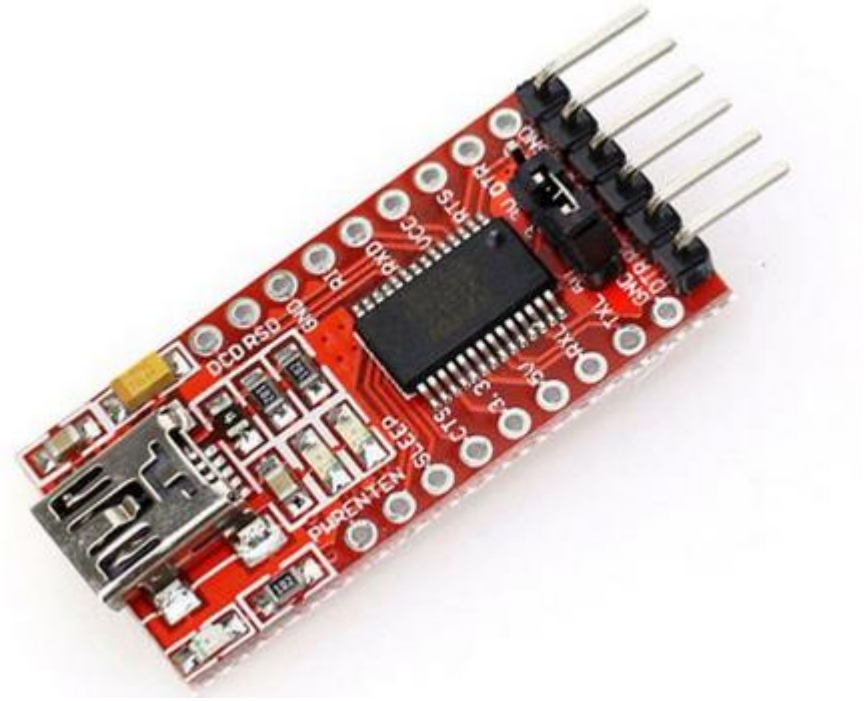
3 Varusohjelmiston luonti

3.1 Ensimmäinen prototyyppi

Tasapainokepin ensimmäinen prototyyppi saatiin käyttöön heti projektin alussa. Ohjain sisälsi Adafruit Pro Trinket -kortin, Adafruit MMA8451 -moduulin, Adafruit Bluefruit LE -piirin ja Adafruit 0,56” näytön [kuva 3].

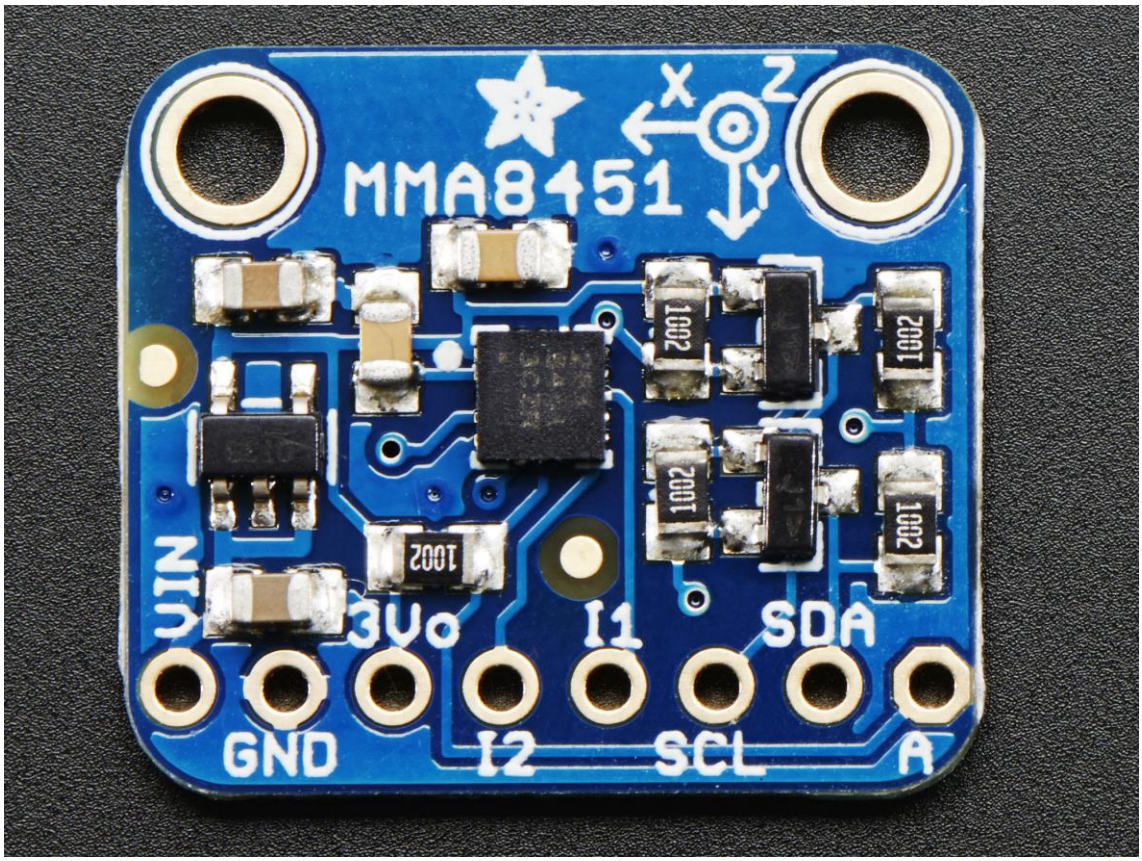
Kuva 3. Ensimmäinen prototyyppiohjain rakennettuna koekytkentäalustalle.

Pro Trinket -kortti sisälsi ATmega328P-prosessorin, jota käytettiin myös toisessa prototyypissä. Kortti toimi 12 MHz:n kellotaajuudella, minkä lisäksi se sisälsi Micro-USB-portin ohjelmiston lataamista varten. Kustannusten pitämiseksi alhaisina Adafruitin korttiin ei ole kuitenkaan lisätty Serial-to-USB-piiriä, joka muuntaisi sarjaliikenteisen tiedonsiirron toimivaksi USB:n välityksellä. [18.] Siten prosessorilta saatavaa dataa jouduttiin lukemaan käyttämällä FT232RL-sarjaliikennemuunninta [kuva 4] ja erillistä pääteemulaattoria (Terminal).



Kuva 4. FT232RL-sarjaliikennemuunnin [19].

Ensimmäisessä prototyypissä käytettiin MMA8451-kiihtyvyysanturia, joka oli integroitu Adafruit MMA8451 -moduuliin [kuva 5]. Anturia käytetään 3 V:n käyttöjännitteellä, joten Adafruit oli lisännyt moduuliin 3,3 V:n regulaattorin [20]. Regulaattorin tarkoituksena on muuttaa korkeampi, mahdollisesti vaihteleva tulojännite matalammaksi, vakaaksi lähtöjännitteeksi [21]. Kiihtyvyysanturissa käytetään TWI-tiedonsiirtoa (Two Wire Interface), joten moduuli sisälsi samalla SCL (Serial Clock Line)- ja SDA (Serial Data Line) -paikan tiedonsiirtoa varten. Lisäksi moduulissa on kaksi keskeytysnastaa, joita ei käytetty ensimmäisen prototyypin yhteydessä. [20.]



Kuva 5. Adafruit MMA8451 -kiihtyvyysanturimoduuli. Kiihtyvyysanturi moduulin keskellä. [20.]

Ensimmäisen prototyypin ohjelmointiin käytettiin Arduino-ohjelmointiympäristöä. Ohjelmointiin käytettiin valmiita kirjastoja. Nämä kirjastot toimivat suoraan ”paketista”, eli esimerkiksi tiedonsiirtoon, näyttöön kirjoittamiseen ja pääte-emulaattorille kirjoittamiseen on olemassa suoraan funktiot. [22.] Ensimmäisen prototyypin ohjelmointi oli enemmänkin valmiiden palikoiden yhdistelyä. Prototyypin yhteydessä jouduttiin jo kuitenkin alkamaan miettiä toteutustapoja eri funktioille.

3.1.1 Kallistuskulman lukeminen

Prototyyppiin tehtiin ohjelmisto, joka lukee kallistuksen väliltä −90 < x < 90. Kallistuksen laskentaan käytettiin kaavaa 1:

$$\alpha = \tan^{-1}\left(\frac{Y}{\sqrt{X^2 \times Z^2}}\right) \times 2\pi \qquad (1)$$

Kaavassa esiintyvät X, Y ja Z ovat kukin kiihtyvyysanturin eri akseleista muodostuvat kiihtyvyydet, $\tan^{-1}$ tarkoittaa arkustangenttia ja 2π muunnoskerrointa radiaaneista asteiksi [23].

Kaava 1 ottaa kantaa vain tasapainokepin (kiihtyvyysanturin) kääntymiseen pituusakselin suhteen eli kepin kallistuksen (roll). Kallistusta laskettaessa otetaan huomioon kaikki kolme akselia, jolloin kepin nyökkäyksellä (pitch), tai tuttavallisemmin kierrolla, ei ole vaikutusta kallistuksen laskentaan. [23.] Tämä on erityisen tärkeää tulosten luotettavuuden kannalta. Esimerkkinä voidaan pitää kepin nostamista pään yläpuolelle, sillä käsien asento tulee muuttumaan kepin ollessa matkalla pelaajan edestä (rystyset eteenpäin) pelaajan yläpuolelle (rystyset ylöspäin). Siten keppi kiertyy luonnollisesti 45^{o}, kun ranteet pidetään suorina. Samalla tasapainokepistä saadaan tehtyä käyttäjäystävällisempi, sillä keppiä käytettäessä näyttöä ei tarvitse pitää ylöspäin vaan toiminta pysyy samanlaisena kierrosta riippumatta.

Keppiin tehtiin myös kallistukseen liittyvä reset-funktio, joka alusti laskurin arvon, kun keppi käännettiin 90^{o}:n kulmaan. Tällöin näytölle tulostettiin arvo 888, joka pysyi siinä kolmen sekunnin ajan. Tämän jälkeen laskurin arvoksi kirjoitettiin 0 ja keppi oli taas käyttövalmiina.

3.1.2 Virhepisteiden laskenta

Virhepisteiden laskenta toteutettiin ensimmäisessä prototyypissä yksinkertaisella kaavalla. Kallistuskulma rajoittui välille $-90 < x < 90$, joten asteikko jaettiin kahdeksaan osaan välillä 0–90 niin, että kallistuksen suunnalla (positiivinen tai negatiivinen) ei ollut merkitystä. Perusajatuksena oli, että kallistuskulman suuruus määrää laskurin arvon kasvamisen: mitä suurempi kallistus, sitä nopeammin laskuria kasvatetaan. Kallistuksen oli siis määrä kiihdyttää laskurin kasvamista.

Laskurin arvo oli double-tyyppinen muuttuja, jota kasvatettiin 0,0625:n kerrannaisilla. Kerrannaiset olivat väliltä 0–8, ja ne määräytyivät kallistuksen mukaan. Syy liukulukujen käyttämiseen oli se, että näin toteutettuna laskuri ei ikinä hypännyt yhdenkään luvun ylitse vaan jokainen luku (0, 1, 2, 3, ...) näytettiin näytöllä. Tämä antoi käyttäjälle paremman mielikuvan kiihtyvästä laskurista.

Laskurin arvo oli tarpeen muuttaa tulostettaessa väliaikaisesti integer-tyyppiseksi eli kokonaisluvuksi, koska laskuri oli tyypiltään liukuluku eikä näytössä ollut mielekästä näyttää murtolukuja. Siten laskuri kasvoi aina nollasta lukuun 999. Laskurin kasvettua lukuun 999 ei laskuria ollut enää järkeä kasvattaa, sillä arvoa ei olisi pystytty näyttämään näytöllä.

3.1.3 Värinämoottoreiden ohjaus

Ensimmäisessä prototyypissä oli kaksi LED-valoa, joiden tarkoituksena oli havainnollistaa värinämoottoreiden toimintaa. Ajan puutteen takia moottoreiden ohjaus toteutettiin ensimmäisessä prototyypissä niin sanotulla päälle/pois-ratkaisulla, yksinkertaisesti sytyttämällä sen puolen LED-valo, jonne tasapainokeppiä oli kallistettu.

Näiden ratkaisujen myötä ensimmäinen prototyyppi oli valmis esiteltäväksi asiakkaalle, joka vaikutti tyytyväiseltä saatuun tulokseen. Tämän jälkeen ryhdyttiin työstämään toista prototyyppiä.

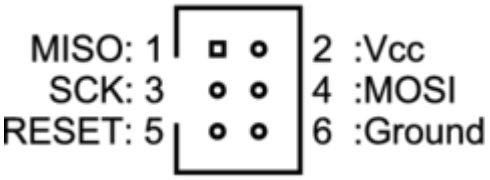
3.2 Toinen prototyyppi

Tasapainokepin toisen prototyypin kohdalla laitteistopuolesta vastaava yritys oli vaihtanut ensimmäisen prototyypin moduulit pelkkiin komponentteihin, joihin Adafruitin moduulit perustuivat. Prosessorina toimi ATmega328P, johon oli liitettynä ulkoinen, 16 MHz:n taajuuden tuottava oskillaattori. Adafruitin MMA8451-moduuli korvattiin Freescale Semiconductorin MMA8451QR1-kiihtyvyysanturilla ja Adafruit Bluefruit LE -moduuli vaihdettiin Lairdin BL600-SA-piiriin.

Toinen prototyyppi oli tarkoitus toteuttaa käyttämällä pelkkiä C-ohjelmointikielen kirjastoja. Tarkoituksena oli hankkiutua eroon Arduinon valmiskirjastoista. Syy muutokseen oli yksinkertaisesti se, että Arduinon kirjastoihin kuuluvat funktiot sisältävät liikaa ylimääräisiä toimia eivätkä ne siten ole tehokkaita. Vanhaa, Arduino-ohjelmointiympäristöllä tehtyä ohjelmaa päätettiin kuitenkin ensin kokeilla uudella prototyypillä. Näin pystyttiin varmistumaan siitä, että toinen prototyyppi toimii samalla tavalla kuin ensimmäinenkin.

3.2.1 Ohjelmointi

Toisessa prototyypissä, kuten ensimmäisessäkin, oli olemassa USB-portti, mutta sitä käytettiin lähinnä piirilevyn virransaantiin ja piirilevyssä kiinni olevan akun lataamiseen. Ohjelmointia varten prototyyppiin oli lisätty ISP-liitäntä (In-System Programming), jonka nastat on aseteltu kuvan 6 mukaisesti.



Kuva 6. ISP-liittimen nastat [24].

Kuvassa on esiteltynä kuusi erilaista nastaa, joista jokaisella on oma käyttötarkoituksensa:

- MISO (Master In, Slave Out): Prosessoriin sisääntuleva liikenne eli prosessoriin yhdistetyn laitteen ulostulo [25].

- V_{cc} (Voltage at the common collector): Nastaan tuleva positiivinen syöttöjännite [26, s. 2].

- SCK (Serial Clock): Kellosignaali, jonka avulla lähettävä ja vastaanottava puoli paritetaan toimimaan samalla taajuudella. Näin molemmat laitteet saadaan synkronoitua, mikä mahdollistaa tiedonsiirron. [25.]

- MOSI (Master Out, Slave In): Prosessorilta lähtevä liikenne eli prosessoriin yhdistetyn laitteen sisääntulo [25].

- RESET: Prosessorin uudelleenkäynnistykseen käytettävä nasta. Laitteistoa ohjelmoitaessa prosessorin reset-linjan pitää olla aktiivisena, jotta siihen voidaan ladata uusi ohjelmisto. RESET-nasta on yhdistetty lähettävän laitteen RESET-nastaan, jotta ohjelmoinnin suorittava laite pystyy ohjaamaan ohjelmoitavaa laitetta. Näin ohjelmoinnin suorittava laite pystyy käynnistämään prosessorin uudelleen niin halutessaan. [26, s. 2.]

- Ground: Maadoitusnasta, yhdistetään prosessoriin yhdistetyn laitteen kanssa yhteiseen maahan [25, s. 2].

Arduino-pohjainen ohjelmisto tarvitsee alkulatausohjelman toimiakseen (bootloader) [27]. Tehtaalta tulevista ATmega328P-prosessoreista se kuitenkin puuttuu, joten prosessoriin ladattiin alustavasti Arduino Nano -ohjelmointialustan alkulatausohjelma. Alkulatausohjelma ladattiin käyttämällä hyväksi erillistä Arduino Nano -korttia ja Arduino-ohjelmointiympäristöä. Arduino Nano yhdistettiin ISP-liitäntään seuraavasti:

- Arduinon D13-nasta (digitaalinen 13, SCK) ISP-liitännän SCK-nastaan (SCK - SCK).

- Arduinon D12-nasta (digitaalinen 12, MISO) ISP-liitännän MISO-nastaan (MISO - MISO).

- Arduinon D11-nasta (digitaalinen 11, MOSI) ISP-liitännän MOSI-nastaan (MOSI - MOSI).

- Arduinon D10-nasta (digitaalinen 10, Slave Select) ISP-liitännän RESET-nastaan (SS - RESET).

- Arduinon 3,3V:n V_{cc}-nasta ISP-liitännän V_{cc}-nastaan (V_{cc} - V_{cc}).

- Arduinon maadoitusnasta (GND) ISP-liitännän Ground-nastaan (GND - GND). [28.]

ATmega328P-prosessori toimii tehtaalta tullessaan 1 MHz:n taajuudella. Ulkoisen kristallin hyödyntämiseksi prosessorin sulakkeita tuli muuttaa niin, että prosessori ymmärsi käyttää ulkoista kristallia. [7; 8.] Sulakkeiden muutos tehtiin alustavasti käyttämällä hyväksi Arduino Nano -ohjelmointialustan asetuksia, jotka siirrettiin prosessoriin Arduino-ohjelmointiympäristön avustuksella.

Arduino-ohjelmointiympäristö pitää kirjaa erilaisista korteista, joita se voi käyttää ohjelmoimisen pohjana. Korttien tiedot sijaitsevat Arduinon kotihakemiston alla tiedostossa boards.txt. Näihin tietoihin sisällytetään muun muassa tiedot käytetyistä taajuuksista ja

jännitteistä sekä sulakkeiden heksakoodi, joka on näin mahdollista siirtää prosessoriin ohjelmoinnin yhteydessä. [28.]

Arduino Nanon käyttäminen ISP-laitteena (In-System Programming) ei ollut natiivisti tuettuna Arduinon puolesta, joten boards-tiedostoon piti lisätä uusi alkio. Alkion tiedot muokattiin vastaamaan Nanon ominaisuuksia. Tämän jälkeen toinen prototyyppi ohjelmoitiin toimimaan Nanon tavoin polttamalla uusi alkulatausohjelma sisälle prosessoriin. [28.] Alkulatausohjelman polttamisen jälkeen prototyyppiä oli mahdollista ohjelmoida käyttämällä Arduino-ohjelmointiympäristöä, ja ensimmäisessä prototyypissä käytetty ohjelmisto oli mahdollista siirtää suoraan uuteen prototyyppiin.

Kun ohjelmistoa käännettiin puhtaasti C-ohjelmointikielelle, siirryttiin käyttämään ohjelmointiympäristönä Atmel Studio 6.2:ta. Näin prosessorin ohjelmointi erillisen Arduino-kortin avulla ei ollut enää mielekästä, vaan ohjelmointi toteutettiin jatkossa käyttämällä ohjelmointiin tarkoitettua Olimex AVR-ISP500 -ohjelmointilaitetta [kuva 7].

Kuva 7. Lopulliseen ohjelmointiin käytetty Olimex AVR-ISP500 -ohjelmointilaite [29].

Laitteessa oli kuvan 7 mukaisesti sekä 6-päinen että 10-päinen ohjelmointiliitin, mutta toisen prototyypin ohjelmointiin käytettiin vain 6-päistä liitintä. Liittimen nastat oli asetettu kuvan 5 mukaisesti, joten laitteen kytkentä oli ennestään tuttua. Ohjelmointilaitteen käyttö helpotti prototyypin ohjelmointia, mutta suurin etu sen käytössä oli, että laitteen kautta päästään suoraan käsiksi prosessorin asetuksiin, toisin sanoen sulakkeisiin.

3.2.2 Sulakkeet

ATmega328P-prosessorissa on käytössä yhteensä 11 erilaista sulaketta (fuse), joilla prosessorin toimintaa on mahdollista säätää laitetasolla. Sulakkeiden avulla prosessori saadaan toimimaan omaan käyttötarkoitukseen sopivalla tavalla [7]. Liitteessä 1 on esitelty kaikki ATmega328P-prosessorissa käytössä olevat sulakkeet.

Arduino Nano -kortin sulakkeet sopivat toisen prototyypin käyttötarkoitukseen hyvin, sillä asetukset asettivat prosessorin käyttämään ulkoista, yli 8 MHz:n oskillaattoria, sallivat ohjelmoinnin sarjaliitännän kautta ja asettivat sallitun käyttöjännitteen minimirajan 2,7 V:iin. Nanon sulakkeiden myötä prosessori käynnistettiin käyttämällä erillistä alkulatausohjelmaa. [30.]

Näistä asetuksista alkulatausohjelman käytön poistaminen oli tarpeellista, sillä puhtaalla C-ohjelmointikielellä kirjoitettuna ja ohjelmointilaitetta käyttämällä ohjelma ladattiin suoraan prosessorin muistiin eikä alkulatausohjelmalle ollut tarvetta [31]. Jänniterajan asettaminen paljasti myös laitteen suunnittelussa puutteen, mutta siihen palataan myöhemmin tämän insinöörityöraportin Lopullinen prototyyppi -luvussa.

3.2.3 TWI-protokolla

Tiedonsiirto ATmega328P-prosessorin ja MMA8451QR1-kiihtyvyysanturin välillä toteutettiin käyttämällä Atmelin tukemaa TWI-protokollaa. TWI tarkoittaa Two Wire Interfacea ja sitä voidaan pitää identtisenä I^2C-protokollan kanssa (Inter-Integrated Circuit) sillä erotuksella, että TWI ei ole rekisteröity tuotemerkki [17].

TWI:ssä tiedonsiirto toteutetaan yhden siirtokanavan ja kellosignaalin avulla, joten tietoa voidaan kuljettaa kerrallaan vain yhteen suuntaan (half-duplex). Tämän takia tiedonsiirto perustuu isäntä-orja-suhteeseen (Master-Slave). Isäntä-osapuoli tarjoaa kellosignaalin, jonka mukaisesti tiedonsiirto suoritetaan. Samalla isäntä-osapuoli aloittaa tiedonsiirron ja määrittelee sen, kumpaan suuntaan tietoa siirretään. Orja-osapuolen tehtävänä on vastata isäntä-osapuolen pyyntöihin. Eri laitteiden tehtävät voivat muuttua tiedonsiirron yhteydessä, ja molempia osapuolia voi olla useita. [1, s. 16–17; 4, s. 207; 32.]

Ennen tiedonsiirron aloittamista on määriteltävä taajuus, jolla tietoa siirretään eri laitteiden välillä. TWI:ssä käytetty kellosignaali muodostetaan käyttämällä hyväksi prosessorin kellosignaalia [4, s. 213]. Prosessoriin on määritelty ennalta kellotaajuuksia, joilla TWI:tä voidaan käyttää. Niitä ovat muun muassa standardi siirtonopeus 100 kHz ja full speed -siirtonopeus, jonka taajuus on suuruudeltaan 400 kHz [32]. Siirtonopeuden laskemiseksi on olemassa kaava 2:

$$TWBR = \frac{\left(\left(\frac{F_{CPU}}{SCL_{CLOCK}}\right)-16\right)}{2} \quad (2)$$

Kaavassa F_{CPU} edustaa prosessorin kellotaajuutta, kun taas SCL_{CLOCK} haluttua tiedonsiirtonopeutta TWI:tä käytettäessä. Projektissa tiedonsiirtonopeus asetettiin 100 kHz:iin. Kaavasta saatu tulos asetettiin prosessorin TWBR-rekisteriin (TWI Bitrate Register), jolloin prosessori osasi asettaa kellosignaalin toimimaan halutulla taajuudella. [4, s. 213.] Kaava on johdettu ATmega328P-prosessorin datalehden sivun 213 kaavasta.

Jokainen TWI:n tiedonsiirtotapahtuma aloitetaan isäntä-osapuolen toimesta lähettämällä aloituskomento START [4, s. 214]. START-komento lähetetään kirjoittamalla prosessorin TWCR-rekisteriin (TWI Control Register) bitit 2 (TWEN, TWI Enable), 5 (TWSTA, TWI Start) ja 7 (TWINT, TWI Interrupt Flag) [4, s. 216]. TWI operoi pitkälti TWCR:n TWINT- ja TWEN-bitin perusteella. Uudet toimet suoritetaan vasta, kun TWI:n keskeytyslippu on laskettu alas kirjoittamalla TWCR:n seitsemänteen eli TWINT-bittiin ’1’. [4, s. 214.] Samaa voidaan sanoa TWEN-bitistä, sillä mitään operaatiota ei suoriteta siirtotiellä ilman, että kyseiseen bittiin on kirjoitettu ’1’ [4, s. 217].

START-komentoa seuraa aina orja-osapuolen laitteen osoite, joka on pituudeltaan 7 bittiä. Osoitteen perään kirjoitetaan START-komennon jälkeen ’0’, joka tarkoittaa sitä, että isäntä-osapuoli tulee kirjoittamaan siirtotielle [4, s. 214]. ’0’ tulee yhdistää aiemmin määriteltyyn laitteen osoitteeseen, eli 7-bittisestä osoitteesta tehdään 8-bittinen lisäämällä nolla sen perään. Esimerkkinä voidaan pitää projektissa käytetyn kiihtyvyysanturin laiteosoitetta 0x1C, joka 7-bittisenä kirjoitetaan muotoon 0011100. Kun perään lisätään nolla, osoitteesta ei suinkaan tule 0x1D vaan 0x38, joka 8-bittisenä kirjoitetaan muotoon 00111000. [1, s. 17–18.]

Seuraavaksi TWI-tiedonsiirrossa siirtotielle kirjoitetaan sen rekisterin osoite, johon dataa halutaan kirjoitettavan tai josta dataa halutaan luettavan [4, s. 214]. Käytettävissä olevien rekistereiden osoitteet löytyvät laitteistojen datalehdiltä. Prosessorin toimiessa isäntä-osapuolena siirtotielle kirjoitetaan halutun orja-osapuolen, tässä tapauksessa MMA8451QR1-kiihtyvyysanturin, rekistereiden osoitteita.

Rekisterin valinnan jälkeen siirtotielle on mahdollista kirjoittaa joko yksi tai useampia tavuja, jolloin haluttu data kirjoitetaan aiemmin määriteltyyn rekisteriin [4, s. 214–215]. Kirjoittamisen lisäksi myös lukeminen on mahdollista aloittaa, mutta lukeminen täytyy aloittaa Repeated Start -prosessin kautta. Prosessin tarkoituksena on lähettää uusi START-komento, kuitenkin niin, että tällä kertaa orja-osapuolen laitteen osoitteen perään lisätään ’1’, joka tarkoittaa sitä, että isäntä-osapuoli tulee lukemaan dataa siirtotieltä aikaisemmin määritellystä orja-osapuolen rekisterin osoitteesta. Tämän jälkeen orja-osapuolen halutusta rekisteristä on mahdollista lukea yksi tai useampia tavuja. [1, s. 17.]

Useiden tavujen kirjoittaminen tai lukeminen on tehty MMA8451QR1-kiihtyvyysanturilla helpommaksi. TWI-prosessia ei tarvitse aloittaa jokaista luku- tai kirjoitusprosessia varten uudestaan vaan riittää, että luettaessa lähetetään pelkkä hyväksyntä tai kirjoitettaessa kirjoitetaan uutta dataa siirtolinjalle. Tällöin kiihtyvyysanturi kasvattaa automaattisesti aiemmin annetun rekisterin osoitetta yhdellä, jolloin seuraava luku- tai kirjoituskomento kohdistuu osoitteeltaan seuraavaan rekisteriin. [1, s. 17–18.] Näin esimerkiksi useiden rekistereiden lukeminen kerralla onnistuu vaivattomasti, ja tällä tekniikalla toteutettiin muun muassa kiihtyvyysantureiden x-, y- ja z-akseleiden rekistereiden MSB:ien ja LSB:ien lukeminen.

Jokaista siirtotielle kirjoitettua komentoa tai dataa seuraa aina ACK-viesti (Acknowledge) tai NACK-viesti (Not Acknowledged). Viestit lähettää aina se osapuoli, joka ei kirjoita siirtolinjalle dataa. ACK-viestin tarkoituksena on lähettää vastapuolelle hyväksyntä siitä, että saatu komento tai data on ymmärretty ja että osapuoli jää odottamaan seuraavaa viestiä. Toisin sanoen jokaista siirtotielle lähetettyä dataa pitää seurata ACK-viesti vastaanottavalta osapuolelta, jotta tiedonsiirtoa voidaan jatkaa. Jos tietoa vastaanottava puoli lähettää NACK-viestin, tietää lähettäjä, että lähetys on joko epäonnistunut tai tiedonsiirto kyseisen osapuolen kanssa halutaan lopettaa ja tiedonsiirtoa ei enää jatketa. [1, s. 17–18; 4, s. 213–215; 32.] Tämä on erityisen tärkeää prototyypin

kannalta, sillä laite ei tule toimimaan ilman kiihtyvyysanturia. Tämän vuoksi funktioihin, jotka aloittavat tiedonsiirron ja määrittävät tiedonsiirrossa käytetyt asetukset, on ohjelmoitu tarkistukset jokaisen siirretyn tiedon yhteyteen.

Jokainen TWI-prosessi lopetetaan lähettämällä STOP-komento [1, s. 17–18]. Komento on muuten samanlainen kuin START-komento, mutta viidennen bitin sijaan (TWSTA) TWCR:ään kirjoitetaan neljäs bitti (TWSTO, TWI Stop) [4, s. 231]. Kuten START-komennon kanssa, myös STOP-komennon yhteydessä tulee TWCR:ään kirjoittaa myös toinen bitti (TWEN) ja seitsemäs bitti (TWINT) [4, s. 216]. STOP-komennon jälkeen siirtolinja isäntä- ja orja-osapuolen välillä katkaistaan [1, s. 17–18; 4, s. 215].

3.2.4 Virhepisteet

Toisen prototyypin yhteydessä virhepistelaskurista haluttiin tehdä dynaamisempi. Laskurin haluttiin reagoivan pieneenkin eroon kallistuksessa sen sijaan, että tietylle astevälille olisi määritelty kiinteä arvo. Virhepisteiden laskenta toteutettiin kaavalla 3, joka muodostettiin ja todettiin toimivaksi käytännön testauksen kautta:

$$Laskuri \mathrel{+}= 0{,}03125 \times \frac{\alpha}{d} \qquad (3)$$

Kaavassa 3 α vastaa kallistuskulmaa ja d integer-tyyppistä vakiota, jonka arvo määritellään ohjelman alustuksessa 10:ksi. Vakiota ei ole kuitenkaan kovakoodattu, eli sen muuttaminen on mahdollista. Muuttaminen suoritetaan erillisellä funktiolla, jota on tarkoitus käyttää Bluetoothin kautta. Funktio ottaa vastaan parametreinä kirjaimen ja kaksi numeroa, joiden avulla vakion d arvoa muutetaan.

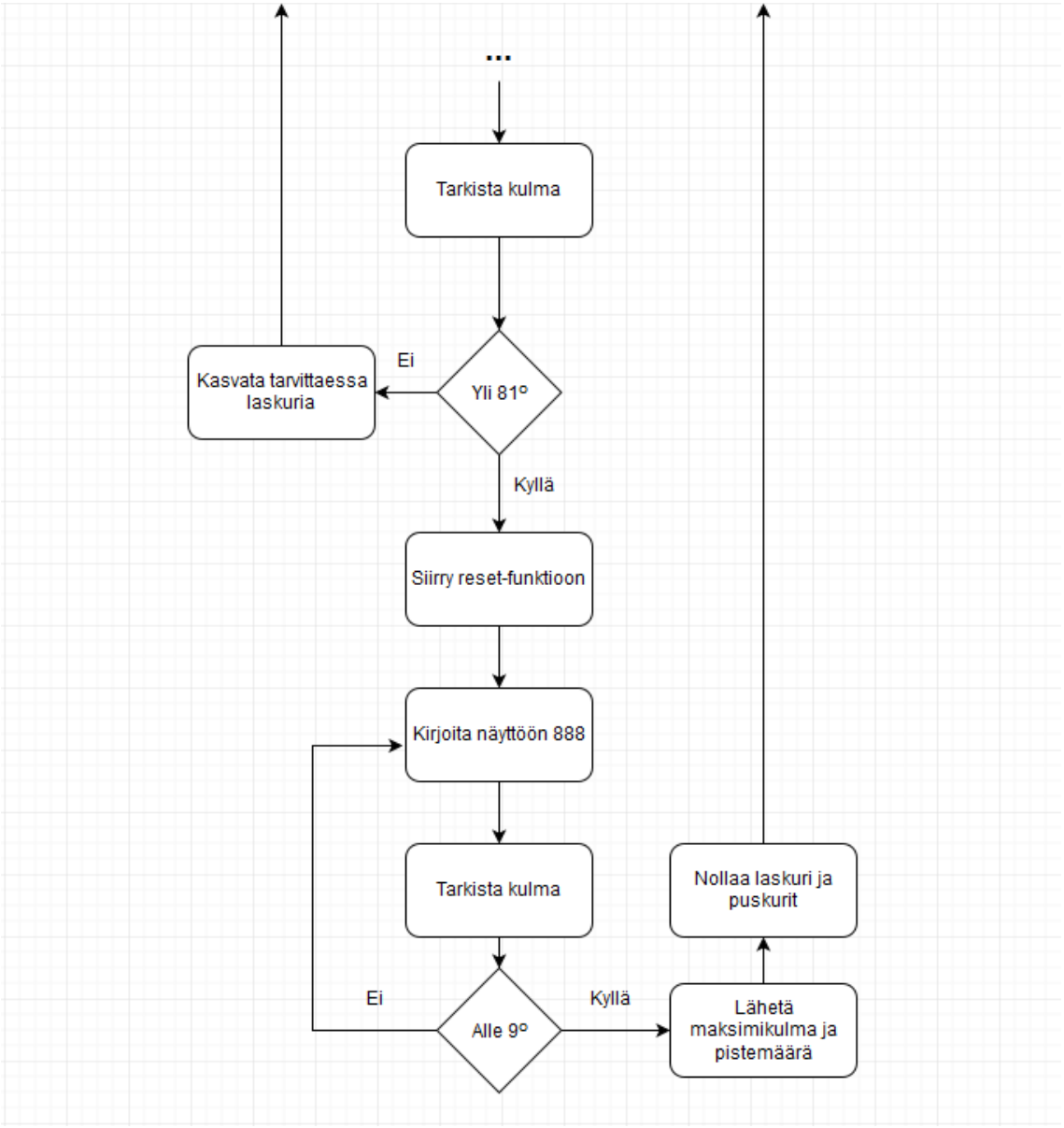
Prototyyppiin koodattiin samalla vakio, joka määrittelee kulman, minkä jälkeen laskurin arvoa lähdetään kasvattamaan. Vakiona tämä kulma on suuruudeltaan 9º, mutta myös sen arvoa on mahdollista muuttaa samaisella funktiolla, jolla aiemmin mainittua jakajaa muutetaan. Täten keppiin pystyttiin toteuttamaan järjestelmä, jolla kepin toimintaan voitiin vaikuttaa ilman, että ohjelmakoodia muutetaan. Tätä funktiota käytetään erityisesti pelin yhteydessä, sillä kulman raja-arvoa ja kaavassa 3 esiintyvää jakajaa säätämällä pelin vaikeusastetta voidaan muuttaa.

Laitteistorajoitteiden myötä pistemäärään kertyi kuitenkin virheitä, sillä tasapainokeppi nollattiin kääntämällä keppi 90º:n kulmaan. Siten laskuri kasvoi jatkuvasti ja sitä jouduttiin kompensoimaan pelin puolelta.

3.2.5 Nollaus

Ensimmäisessä prototyypissä toteutettua reset-funktiota ryhdyttiin vielä parantamaan toista prototyyppiä varten. Perusajatus kepin alustamisessa pysyi ennallaan, mutta nollauksesta pyrittiin tekemään käyttäjäystävällisempi. Ajatuksena oli, että tasapainokeppi nollaantuu vasta sen jälkeen, kun se käännetään takaisin tilaan, jossa pistelaskuria ei kasvateta. Tämä muutos tehtiin pollauksen avulla. Pollauksella eli kiertokyselyllä tarkoitetaan tapaa, jossa ohjelma jää pyörimään paikalleen odottaen haluttua syötettä, tässä tapauksessa kepin tasapainoasemaa [33].

Kun keppi käännetään yli 81 asteen kulmaan, näyttöön kirjoitetaan 888 ja prosessori jää odottamaan käyttäjää. Kun keppi käännetään takaisin alle 9 asteen kulmaan, joka on ilman erillisiä muutoksia alin laskuria kasvattava raja-arvo, keppi asettaa laskurin arvon nollaan ja palauttaa kepin toiminnan ennalleen. Reset-funktion toiminta on esitettynä kuvassa 8, joka demonstroi toiminnan vakioarvoilla esitettynä.



Kuva 8. Tasapainokepin toisen prototyypin reset-funktion toiminta.

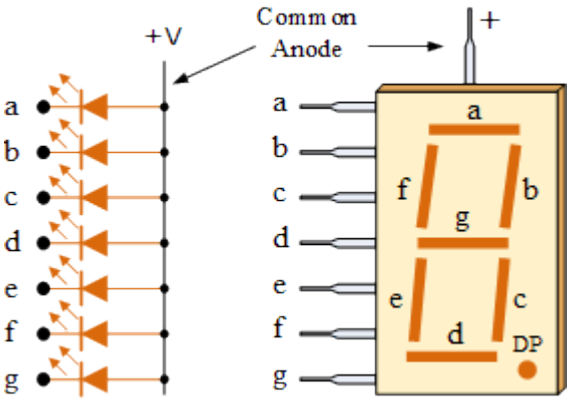
Pollauksella tehty nollaus ei ole virrankulutuksen kannalta tehokkain vaihtoehto, mutta sitä pidettiin prototyyppiin toimivana ratkaisuna. Reset-funktio yritettiin myös toteuttaa laittamalla prosessori unitilaan. Valitettavasti prototyyppilevyjen määrä oli rajallinen ja yhden prototyypin rikkouduttua kesken testien tultiin siihen päätökseen, että unitilan toteuttaminen ei välttämättä ollut toimiva ratkaisu eikä sitä uskallettu yrittää uudelleen projektin keskeytymisen pelossa.

3.2.6 Näytöt

Projektissa käytettiin 7-segmenttinäyttöjä, jotka oli yhdistetty suoraan ATmega328P-prosessorin nastoihin. Siten näyttöjen ohjaaminen oli tehty tehokkaaksi ja helpoksi toteuttaa suoraan prosessorilta käsin.

Näyttöjen toimintaperiaate on yksinkertainen. Nimensä mukaisesti 7-segmenttinäyttö on jaettu seitsemään erilliseen osaan, jotka oli tässä tapauksessa nimetty kirjaimilla A-G, kuten kuvassa 9. Näytön jokainen osa on itsenäinen yksilö eikä näin ollen ole riippuvainen toisten osien tiloista. Näytöt on yhdistetty referenssipisteeseen, jonka mukaisesti näytöt toimivat: jos yksittäisen LED-valon, joka on tässä tapauksessa yhdistetty prosessorin nastaan, ja referenssipisteen välillä on jännite-ero, virta kulkee LED-valon läpi ja kyseinen osa näytöstä syttyy. [34.]

Jännite-ero muodostetaan referenssipisteestä esimerkiksi pisteeseen A, kun käytetään kuvan 9 mukaisia näyttöjä, koska LED-valot johtavat virtaa vain toiseen suuntaan [35]. Toisin sanoen näytön ohjaaminen on mahdollista niin, että yhteiseen referenssipisteeseen syötetään jännite, minkä jälkeen näytön eri osat vedetään maihin (nollajännitteeseen) sen mukaisesti, mikä numero halutaan saada näytölle näkyviin. Näytön sammuttaminen taas onnistuu vetämällä yhteinen referenssipiste maihin, jolloin positiivista jännite-eroa referenssipisteestä nimettyyn pisteeseen ei ole olemassa. [34.]

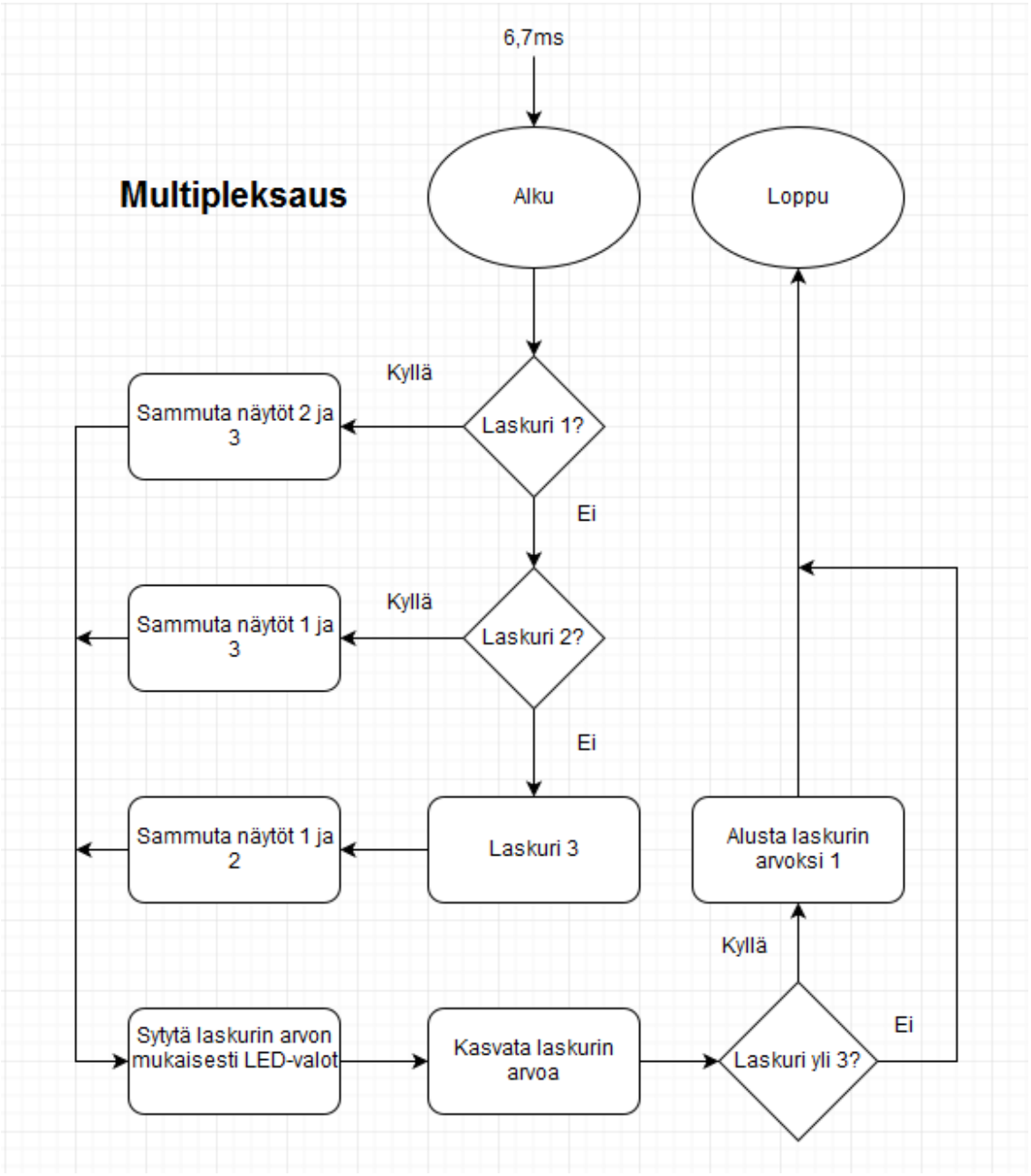


Kuva 9. 7-segmenttinäyttöjen toimintaperiaate [34].

Tasapainokepin toteutuksessa käytettiin kolmea erillistä 7-segmenttinäyttöä, jotka oli yhdistetty kolmeen erilliseen referenssipisteeseen. Jokaiselle näytölle ei ollut varaa antaa omia nastoja näyttöjen ohjaamiseksi, koska prosessorin nastojen määrä oli rajal-

linen. Tästä syystä samalla kirjaimella nimetyt osat oli yhdistetty samaan prosessorin nastaan. Näyttöjen oli tarkoituksena toimia laskurina, joka näyttää pelaajan virhepisteiden määrän alkaen nollasta ja päättyen lukuun 999. Siten näyttöjen ohjaaminen yhtäaikaisesti ei ollut mahdollista, sillä yhtäaikaisesti ohjattuna näytöt näyttäisivät kaikki samaa lukua (000, 111, 222, ...) eikä laskurin toteutus onnistuisi.

Näyttöjen ohjaus ratkaistiin niin sanotulla multipleksauksella [kuva 10]. Multipleksauksen tarkoituksena on kuljettaa useita signaaleita samaa kanavaa pitkin, jolloin kanavasta on mahdollista valita haluttu signaali kullakin ajan hetkellä [36]. Toisin sanoen näyttöjen ohjaaminen oli pakko toteuttaa kolmena erillisenä toimenpiteenä. Näyttöjä ohjattiin jatkuvassa kierrossa, eli jokaista näyttöä ohjattiin vuorollaan.



Kuva 10. Näyttöjen ohjaus multipleksauksella.

Näyttöjen ohjaukseen otettiin käyttöön ajastettu TIMER0-laskuri ja vastaava keskeytysvektori. Keskeytyksen toimintataajuudeksi asetettiin 150 Hz, eli keskeytys tapahtui noin 6,7 millisekunnin välein, ja keskeytyksen sisälle laitettiin erillinen, staattinen laskuri. Laskurin tarkoituksena oli pitää kirjaa siitä, mitä näyttöä oli kulloinkin määrä ohjata. Näin jokaista näyttöä oli mahdollista ohjata vuorotellen.

Jokaisen näytön kohdalla toimittiin samalla tavalla. Keskeytysvektorissa tarkistettiin aina vuorossa olevan näytön numero. Tämän jälkeen muut näytöt sammutettiin ja vuorossa olevaan näyttöön kirjoitettiin haluttu arvo. Arvo laskettiin jokaiselle näytölle erilaisella kaavalla, ja laskenta perustui virhepistelaskurin antamaan arvoon. Täten sadat, kymmenet ja ykköset saatiin jaettua virhepistelaskurista omiin näyttöihinsä. Kun arvo oli kirjoitettu näyttöön, kasvatettiin näyttölaskurin arvoa ja siirryttiin seuraavaan näyttöön. Keskeytyksen toimintataajuuden ollessa riittävän suuri ihmissilmä ei huomaa näyttöjen sammuttamista ja käynnistämistä. Siten näytöt näyttävät palavan jatkuvasti, vaikka todellisuudessa ne vilkkuvat 50 Hz:n taajuudella. Keskeytysvektorin toiminta näkyy kuvassa 10.

3.2.7 Värinämoottorit

Tasapainokepin toiseen prototyyppiin tehty moottorinohjaus toteutettiin keskeytysten avulla. Ensimmäisessä prototyypissä moottoreiden ohjaus toteutettiin vain päälle/pois-mekanismilla, mutta asiakkaan toiveesta moottorit muutettiin värähtelemään 100 millisekunnin pituisilla pulsseilla.

Moottoreiden toimintaa ohjattiin TIMER1-laskurilla ja vastaavalla keskeytysvektorilla. Laskuri alustettiin tasapainokepin käynnistyksen yhteydessä. Laskuri toimii nimensä mukaisesti ajastettuna, eli alustamisen yhteydessä määriteltiin taajuus, jolla keskeytysvektori toimii. 100 millisekunnin pulssit saatiin määrittelemällä taajuus kaavan 4 mukaisesti:

$$f = \frac{1}{T} \tag{4}$$

Kaavaan sijoitettiin muuttujan T paikalle 100 ms, jolloin vektorin käyttötaajuudeksi saatiin 10 Hz.

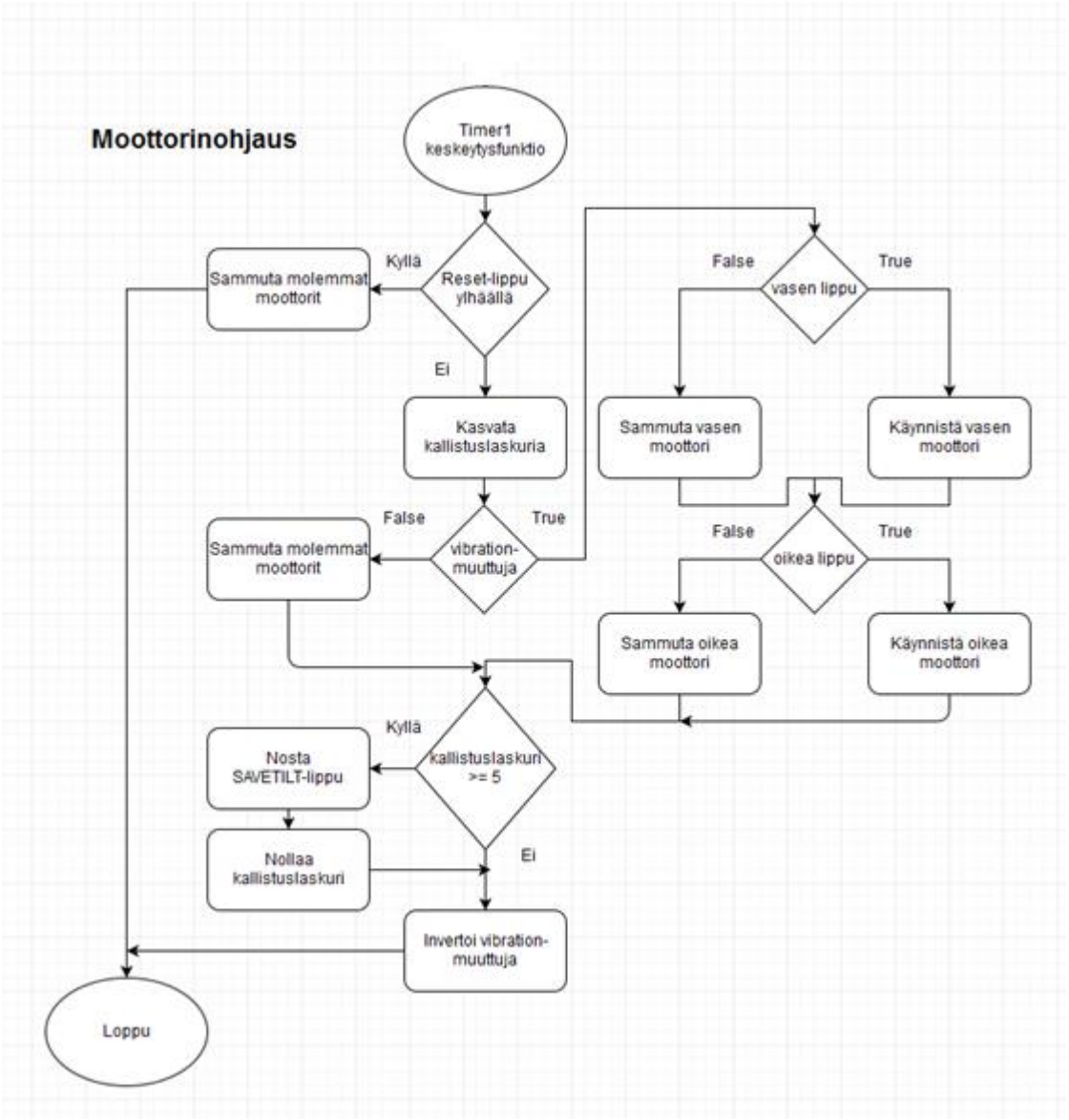
Syy TIMER1-laskurin käyttöön oli, että se on kolmesta ATmega328P-prosessorissa olevasta laskurista (TIMER0, TIMER1, TIMER2) ainoa, jonka laskuri on 16-bittinen. Muut laskurit rajoittuvat 8-bittisyyteen [4, s. 1]. Tällä oli merkitystä taajuutta määriteltäessä, sillä laskurit TIMER0 ja TIMER2 eivät päässeet riittävän alhaisiin taajuuksiin. Taajuus määritettiin kirjoittamalla prosessorin OCR1A-rekisteriin kaavasta 5 saatava arvo:

$$OCR1A = \left(\frac{F_{CPU}}{(f \times PRE)}\right) - 1 \quad (5)$$

Kaavassa F_{CPU} on prosessorin käyttötaajuus, f on keskeytykselle haluttu taajuus ja PRE on niin sanottu prescaler [4, s. 123]. Yksinkertaistettuna prescaler määrittelee sen, kuinka monta kierrosta prosessori tekee, ennen kuin laskurin arvoa kasvatetaan yhdellä. Esimerkiksi prescalerin ollessa 256 laskuri odottaa 256 prosessorin kierrosta, ennen kuin laskuri kasvattaa arvoaan yhdellä. Kaavasta 5 lasketun arvon tulee olla alle 2^{16} = 65536, joka on 16-bittisen laskurin suurin sallittu luku. Kaavaan on määritelty −1, koska rekisterissä laskenta aloitetaan nollasta. [37.]

Laskurin toiminta on yksinkertainen: laskuria kasvatetaan prescalerin ja prosessorin kierrosten määräämään tahtiin. Kun laskuri saavuttaa raja-arvon, joka on määriteltynä OCR1A-rekisteriin, laukaisee laskuri keskeytysvektorin TIMER1_COMPA_vect ja alustaa laskurin takaisin nollaan [4, s. 119; 4, s. 122–123].

Itse keskeytysvektori toteutettiin moottorinohjauksen osalta kolmen erillisen lipun avulla. Näistä yksi lippu oli tarkoitettu kutakin moottoria kohden ja yksi lippu oli varattu nollaukselle, minkä lisäksi käytettiin boolean-muuttujaa vibration. Nollaukseen tarkoitetun Reset-lipun ollessa ylhäällä molemmat moottorit sammutetaan ehdoitta. Molemmilla moottoreilla on omat lippunsa, eivätkä ne voi olla yhtä aikaa aktiivisia. Pulssitus tehtiin vibration-muuttujan avulla muuttamalla muuttujan tilaa jokaisella kierroksella vastakkaiseksi, kuten kuvasta 11 voi nähdä.

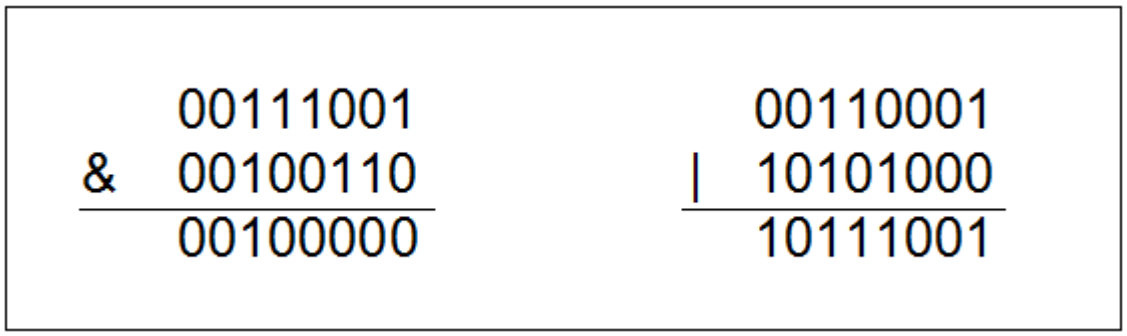


Kuva 11. Moottorinohjaukseen käytetty keskeytysfunktio.

Keskeytysrutiinissa suoritettavat moottoreiden ohjaukset on toteutettu bittioperaatioina. Syy bittioperaatioiden käyttöön on yksinkertaisesti tehokkuus: keskeytyksessä tehtävät asiat on pidettävä mahdollisimman lyhyinä [38].

Moottoreiden sammutus toteutettiin AND-funktion (JA) avulla, kun taas moottoreiden käynnistykseen käytettiin OR-funktiota (TAI). Molemmissa käytettiin apuna maskausta. Esimerkiksi 8-bittisen luvun maskauksella tarkoitetaan apuna käytettävää kahdeksaa bittiä, joiden avulla AND- ja OR-funktiot toteutetaan. AND- ja OR-funktioiden toiminta on nähtävissä kuvassa 12: AND-funktiossa molempien verrattavien bittien tulee olla ’1’,

jotta lopulliseen tulokseen tuleva bitti on ’1’, kun taas OR-funktiossa riittää, että toinen biteistä on ’1’, jotta lopputulokseen tuleva bitti on ’1’.



Kuva 12. Esimerkit bittioperaatioista AND ja OR.

Näin tekemällä saadaan hallitusti muutettua moottoreiden tilaa. Moottoreita sammutettaessa käytetään maskia, joka sisältää pelkkiä ykkösiä lukuun ottamatta sammutettavan moottorin bittiä. Moottoreita käynnistettäessä käytetään maskia, joka sisältää pelkkiä nollia lukuun ottamatta käynnistettävän moottorin bittiä. Siten moottorit saadaan sammutettua ja käynnistettyä ilman, että vaikutetaan tasapainokepin muuhun toimintaan.

3.2.8 UART-kommunikaatio

UART-funktiot (Universal Asynchronous Receiver/Transmitter) rakennettiin aluksi vianetsintään ja tiedonsiirtoon tasapainokepin ja tietokoneen välille, tarkoituksena korvata Arduinon Serial-komennot. Loppujen lopuksi UART-funktioita käytettiin kuitenkin myös BLE-moduulin ja sitä käyttävän laitteen väliseen tiedonsiirtoon. UART-funktiot toteutettiin käyttämällä pohjana valmista esimerkkiä UART:n toiminnasta [39].

UART-tiedonsiirron perustana on tiedonsiirtonopeuden eli baudinopeuden laskenta [39]. Atmel Studiossa oli valmiiksi määriteltynä makro (funktio), jota käyttämällä baudinopeudet oli mahdollista laskea, mutta makroa käytettäessä pääte-emulaattorille tulostui lähinnä epämääräisiä merkkejä. Tämän takia nopeuksien laskemiseksi päädyttiin käyttämään kaavaa 6, josta saatu arvo kirjoitettiin kahdessa osassa 8-bittisiin rekistereihin UBRR0H ja UBRR0L, jotka ohjaavat UART:n tiedonsiirtonopeutta:

$$BAUD_{REG} = \left(\frac{F_{CPU}}{16 \times V_{BAUD}}\right) - 1 \quad (6)$$

Kaavassa F_{CPU} on prosessorin kellotaajuus ja V_{BAUD} on haluttu baudinopeus. Kaavasta saatava arvo voi olla parhaassa tapauksessa 12-bittinen, joten UBRR0H-rekisteriin kirjoitettiin neljä ylintä bittiä siirtämällä $BAUD_{REG}$-muuttujaan saatua bittiarvoa kahdeksan pykälää oikealle ($BAUD_{REG}$ >> 8) ja UBRR0L-rekisteriin kahdeksan alinta bittiä kirjoittamalla siihen $BAUD_{REG}$-muuttujan arvo. [40.]

Jos haluttu baudinopeus asettuu sellaiselle alueelle, että prosessori ei pysty sitä suoraan tukemaan, ohjelmoidaan UART toimimaan kaksinkertaisella nopeudella nostamalla UCSR0A-rekisterin (UART Control and Status Register A) U2X-bitti ylös [39]. Tiedonsiirtoon käytettiin baudinopeutta 9600.

Alustuksen lopuksi sarjaliikenteeseen tulee määritellä siirrettävän paketin koko. Paketin koko määriteltiin kahdeksaan bittiin eli yhteen tavuun, jota seuraa aina yksi STOP-bitti. Pakettikoko määriteltiin kirjoittamalla UCSR0C-rekisteriin UCSZ01- ja UCSZ00-muuttujan (UART Character Size) bittiarvot. Lopuksi tiedonsiirto aktivoidaan molempiin suuntiin kirjoittamalla UCSR0B-rekisteriin bittiarvot muuttujista RXEN0 (Receiver enable) ja TXEN0 (Transmitter enable). [40.]

Jotta UART saadaan vastaamaan käyttäjän antamiin syötteisiin, tulee tiedonsiirtoon osallistuvan laitteen STDIO-virta (Standard Input/Output) ohjata käyttämään UART:a. Ohjaus saadaan tehtyä Atmel Studion makrolla FDEV_SETUP_STREAM (put, get, rwflag), joka ohjaa IO-virran makron put- ja get-parametriin määriteltyihin funktioihin. Makron rwflag-parametriin kirjoitetaan tiedonsiirron suunta eli luku tai kirjoitus. Siten makroa tuli käyttää kahteen kertaan: kertaalleen niin, että makrolle annettiin put-parametri ja kirjoitusta tarkoittava arvo ja kertaalleen taas get-parametri ja lukemista tarkoittava arvo. [39.]

Sarjaporttiin tulevaa liikennettä luettiin käyttämällä char uart_getchar (FILE *stream) -funktiota, joka lukee yhden merkin kerrallaan ja palauttaa sen. Funktio on suhteellisen yksinkertainen, sillä se jää odottamaan syötettä käyttämällä makroa loop_until_bit_is_set (UCSR0A, RXC0). Makrolle siis annetaan tieto siitä, minkä rekisterin mitäkin bittiä tulee seurata. Tässä tapauksessa UCSR0A-rekisterin RXC0-bitti nousee ylös, kun dataa on saatavilla. Kun bitti nousee pystyyn, palautetaan UDR0-

rekisterissä (UART Data Register) oleva merkki sitä pyytäneelle funktiolle, jolloin RXC0-bitti laskeutuu takaisin alas. [39.]

Sarjaporttiin kirjoitettiin lähestulkoon samalla tavalla. Kirjoittamiseen käytettiin funktiota void uart_putchar(char c, FILE *stream). Funktion alussa tarkastetaan onko saatu merkki rivinvaihto, sillä itsenäisesti toimiessaan rivinvaihto vaihtaa kohdistimen paikkaa vain yhtä alemmas sen sijaan, että palauttaisi sen rivin alkuun. Tämän takia rivinvaihdon yhteydessä funktion tulee kutsua itse itseään rekursiivisesti käyttämällä ’\r’-merkkiä eli carriage returnia (vaununpalautus). Tarkistuksen jälkeinen toiminta muistuttaa lukemista, kuitenkin sillä erotuksella, että loop_until_bit_is_set-makro seuraa tällä kertaa TXC0-bitin toimintaa ja UDR0-rekisteriin kirjoitetaan merkki palauttamisen sijaan. [39.]

UART:n kautta tehty sarjaliikenne, jota käytettiin Bluetoothin yhteydessä, toteutettiin lukemisen osalta käyttämällä ATmega328P-prosessorin sisäistä keskeytysvektoria USART_RX_vect. Keskeytys aktivoitiin kirjoittamalla UCSR0B:n rekisteriin RXCIE0-muuttujan (UART Receive Complete Interrupt Enable) bittiarvo. Tällöin keskeytys tapahtuu aina, kun ATmega328P-prosessorin rekisterissä UDR0 on jotain dataa eikä sitä ole luettu. [4, s. 181–182.] Näin Bluetoothin kautta saatiin lähetettyä tehokkaasti dataa ilman, että mitään hukkui matkalla. Tämä oli tärkeää tiedonsiirron eheyden takaamiseksi, sillä lukemista toteutettaessa huomattiin, että esimerkiksi ajoitetun keskeytyksen nopeus ei riittänyt lukemaan kaikkea Bluetoothin kautta lähetettyä dataa.

Sarjaliikenteen pohjimmaisena tarkoituksena oli lähettää tasapainokepiltä saadut tulokset pelille prosessoitavaksi, mutta keppiin haluttiin myös toteuttaa funktio, jolla pystyttiin muuttamaan suoraan, Bluetoothin välityksellä, keppiin asetettuja arvoja. Täten keppiin liittyvästä pelistä olisi mahdollista ohjata tasapainokepin toimintaa.

Keppiin määriteltiinkin funktio, jolla asetusten muuttaminen oli mahdollista. Muutettavia arvoja olivat kallistuksen raja-arvo, jolla virhepistelaskuri alkaa laskea pisteitä, minkä lisäksi virhepisteiden laskentakaavaan liittyviä arvoja on mahdollista muuttaa. Siten erilaisten vaikeusasteiden toteuttaminen on mahdollista pelistä käsin. Samalla kepin nollaus pystyttiin toteuttamaan langattomasti, joten jatkossa kepin toimintaa pystytään ohjaamaan ilman, että kepin lähdekoodia muutetaan.

Tietoa voidaan siirtää myös tasapainokepiltä pelille. Jokaisen reset-funktion yhteydessä laskurin silloinen arvo lähetetään Bluetoothin välityksellä, jolloin peli osaa laskea saadut pistemäärät ja toteuttaa pelin päässä omat funktionsa. Näin peli pystyy näyttämään virhepisteiden määrän sekä pelissä että tasapainokepissä, minkä lisäksi muun muassa maksimaalinen kallistuskulma saadaan näytettyä pelin pelaajalle.

3.2.9 Bluetooth

Bluetoothin toteuttamiseksi käytettiin Laird Technologiesin BL600-SA-moduulia. Moduuli oli BLE-tyyppinen (Bluetooth Low Energy) eli se toimii alhaisilla virtatasoilla. BLE-tyyppistä moduulia käytetään tyypillisesti niin, että moduuli aktivoituu tietyin väliajoin, lähettää tiedot ja vaipuu taas ”horrokseen” [9]. Projektissa moduulia käytettiin niin, että esimerkiksi pelin vaikeusasteen säätäminen onnistuu pelistä käsin, minkä lisäksi kepistä lähetettiin jokaisen tehdyn liikkeen päätteeksi tietoa liikkeen onnistumisesta.

Yhteyden muodostamiseksi BL600-SA-moduulin ja laitteen välille käytettiin yleisesti hyväksyttyä GATT-profiilia (Generic Attribute Profile), joka määrittelee sen, kuinka kaksi BLE-laitetta keskustelevat keskenään eli mitä palveluita ja piirteitä käytetään tiedonsiirtoon. GATT määrittelee käytettäväksi GATT-palvelimen (GATT-server) ja GATT-asiakkaan (GATT-client). Palvelin sisältää käytettävät palvelut ja piirteet, kun taas asiakas lähettää tietopyynnöt palvelimen päähän. Kaikki tiedonsiirto perustuu asiakkaaseen, joka toimii GATT-profiilin yhteydessä niin sanottuna isäntä-laitteena (Master device). Palvelimen tehtävänä on vastata asiakkaan tekemiin pyyntöihin, eli palvelin toimii tässä tapauksessa niin kutsuttuna orja-laitteena (Slave device). [9; 41.]

Yhteys moduulin ja Unityn välille muodostettiin käyttämällä BL600-SA-moduulissa mukana tulevaa virtuaalisarjaporttiprotokollaa [16]. Protokollan sisältämät palvelut on identifioitu käyttämällä UUID-merkkisarjoja (Universally Unique Identifier), joiden perusteella jokainen palvelu voidaan yksilöidä yhden merkkijonon taakse [42]. Palveluita voidaan verrata esimerkiksi prosessoreissa käytettyihin rekistereihin: tietyn UUID:n takana on tietynlainen palvelu, jota voidaan käyttää hyväksi oman ohjelman tuottamiseksi. Virtuaalisarjaporttiin liittyvät palvelut on määritelty taulukossa 1.

Taulukko 1. Virtuaalisarjaportin palvelut ja palveluita vastaavat UUID:t [16].

PALVELU	**UUID**
TX FIFO	569a**2000**-b87f-490c-92cb-11ba5ea5167c
RX FIFO	569a**2001**-b87f-490c-92cb-11ba5ea5167c
ModemOut	569a**2002**-b87f-490c-92cb-11ba5ea5167c
ModemIn	569a**2003**-b87f-490c-92cb-11ba5ea5167c

Ideana on, että asiakas ja palvelin keskustelevat käyttämällä kyseisiä UUID:itä. Kun asiakas kirjoittaa ModemIn-palveluun osoittavaan osoitteeseen merkin ’1’, palvelin tietää, että asiakas on valmis ottamaan palvelimen lähettämää dataa vastaan ja siirtokanava asiakkaan ja palvelimen välille avataan. Tämän jälkeen palvelimelta lähetetään dataa käyttämällä TX FIFO -palvelun osoitetta. Datan lähettäminen on mahdollista niin kauan, kunnes asiakkaalta lähetetään ModemIn-palvelun osoitteeseen merkki ’0’, mikä tarkoittaa sitä, että asiakaslaite ei enää kuuntele kyseistä osoitetta. [16.]

Vastaavasti tiedonsiirto asiakaslaitteelta palvelimelle mahdollistetaan käyttämällä hyväksi ModemOut- ja RX FIFO -palvelua. Kun palvelin kirjoittaa ModemOut-palveluun arvon ’1’, se ilmoittaa asiakkaalle, että dataa voidaan siirtää palvelimen suuntaan. Tämän jälkeen tiedot lähetetään asiakkaan toimesta RX FIFO -palvelun osoitteeseen. [16.]

Käytännössä datan lähettäminen ja vastaanottaminen TX FIFO - ja RX FIFO -palvelun kautta toteutettiin hieman eri tavalla. Peli, johon keppi yhdistettiin, laitettiin kuuntelemaan Bluetooth-moduulin (palvelin) TX FIFO -palvelun osoitetta käyttäen takaisinkutsu-funktiota (callback). Kun puskuriin ilmestyi dataa, ilmoitti takaisinkutsu-funktio siitä pelille. Tämän jälkeen peli (asiakas) lähetti Bluetooth-moduulin ModemIn-palvelun osoitteeseen merkin ’1’ ja otti vastaan ennalta määritellyn merkkimäärän. Kun kaikki merkit oli saatu talteen, lähetettiin ModemIn-palvelun osoitteeseen merkki ’0’, jolloin tiedonsiirto lopetettiin. Saadun datan käsittely toteutettiin pelin päässä, eikä se ole oleellista tämän insinöörityön kannalta.

Datan lähettäminen peliltä kepille oli yksinkertaisempaa. Peli yksinkertaisesti lähetti dataa Bluetooth-moduulin RX FIFO -palvelun osoitteeseen, kun sille oli tarvetta. Kun dataa oli lähetetty osoitteeseen, siitä tuli tieto prosessorille, sillä Bluetooth-moduuli oli yhteydessä prosessoriin. Tällöin prosessorin keskeytysvektorin USART_RX_vect lippu

nousi pystyyn ja prosessori pystyi lukemaan saadut datat suoraan Bluetooth-moduulista käyttämällä UART:a [4, s. 181–182]. Näin dataa voitiin lähettää kepille ajankohdasta riippumatta, jolloin keppi toimi dynaamisesti eikä ohjelman suorittamista tarvinnut keskeyttää säännöllisesti mahdollisen tiedonsiirron takia.

3.2.10 Datapuskurit

Bluetoothilta saatavan datan tallettamiseen käytettiin yksinkertaista kahdeksan merkin kokoista char-taulukkoa. Taulukon toiminta rakennettiin jonomaisesti eli FIFO-periaatteella (First In, First Out). Näin Bluetoothin kautta pystyttiin lukemaan kokonaisia merkkijonoja, jotka käsiteltiin ohjelmassa ensimmäisenä tulleen merkin mukaisesti.

Jono toteutettiin kahden apumuuttujan, uint8_t first ja int8_t last, avulla, joilla nimen mukaisesti pidettiin kirjaa puskurissa olevien merkkien sijainneista: first viittaa taulukon ensimmäisen alkion paikkaan ja last viimeiseen alkioon, joka taulukkoon on tuotu. Last oli tyypiltään int8_t, sillä tyhjässä taulukossa muuttuja osoittaa paikkaan −1. Jono toteutettiin rengaspuskurina: viimeiseen paikkaan tallentamisen jälkeen siirryttiin takaisin ensimmäiseen paikkaan.

Puskurista lukeminen oli mahdollista niin kauan, kuin puskurissa riitti merkkejä. Puskuriin luettava alkio siirrettiin apumuuttujaan, minkä jälkeen alkio tyhjennetään ja ensimmäisen alkion sijaintiin viittaavaa first-muuttujaa kasvatetaan yhdellä. Tämän jälkeen palautetaan puskurista luettu arvo. Jos puskurista lukeminen ei onnistu, palauttaa funktion merkin ’Q’.

Jokaisen merkkijonon käsittelyn jälkeen puskuri tyhjennettiin käyttämällä funktiota, joka kävi jokaisen puskurin alkion lävitse ja alusti alkion nollaksi. Tämän jälkeen muuttujat first ja last laitettiin osoittamaan takaisin alkuperäisiin sijainteihinsa eli nollaan ja −1:een.

Tämän lisäksi tasapainokeppiin tehtiin toinen, 100-paikkainen uint_8-tyyppinen rengaspuskuri, johon tallennettiin kiihtyvyysanturilta saatuja kallistuskulmia. Puskurin tarkoituksena oli määrittää maksimaalinen kallistuskulma, johon keppi oli käännetty yhden pelissä esiintyvän kentän aikana.

Kallistuskulman lukeminen toteutettiin värinämoottoreihinkin käytetyn TIMER1-funktion yhteydessä. Kulma luettiin kaksi kertaa sekunnissa, minkä lisäksi puskurin täyttyessä pienimpiä arvoja ryhdyttiin poistamaan puskurista.

Maksimaalista kulmaa määritettäessä huomattiin sama ongelma kuin virhepistelaskurin yhteydessä: nollauksen ollessa tasapainokepin kääntö 90^{o}:n kulmaan, oli tulosten vääristyminen mahdollista. Tämän vuoksi reset-funktion yhteydessä viisi viimeksi tullutta kallistuskulmaa poistettiin puskurista. Toisin sanoen keppiä oli täten aikaa kääntää 2,5 sekuntia liikkeen lopettavaan asentoon. Kulman määrityksen jälkeen puskuri tyhjennettiin kaikista arvoista.

4 Lopullinen prototyyppi

4.1 Ongelmat

Erityisesti tasapainokepin toisen prototyypin valmistuksessa kohdattiin erinäisiä ongelmia, jotka ratkaistiin mahdollisuuksien mukaisesti. Jokaisesta ongelmasta ilmoitettiin asiakkaalle, ja tarvittavat korjaukset tehdään lopulliseen, tuotantoon menevään tuotteeseen.

4.1.1 Jännitetasojen epävakaus

Toinen prototyyppi ohjelmoitiin käyttämään Arduino Nanon alkulatausohjelmaa, joka oli käytössä myös ensimmäisessä prototyypissä. Oletettiin, että ohjelmiston siirtäminen laitteesta toiseen olisi ollut vaivatonta, koska kaikki komponentit olivat pohjimmiltaan samoja. Ohjelmisto ei kuitenkaan toiminut toivotulla tavalla toisessa prototyypissä: kun dataliikennettä seurattiin sarjaportin kautta, ei prosessori syöttänyt ulos mitään järkevää.

Erilaisten vianselvitysyritysten jälkeen siirryttiin tutkimaan itse prosessorin toimintaa, sillä kaikki muu vaikutti olevan kunnossa. Erillisen vianselvityslaitteen avulla kuitenkin nähtiin, että noin kahden millisekunnin välein prosessori sammutti ja käynnisti itsensä uudelleen. Uudelleenkäynnistyminen jatkui loputtomassa kierrossa, ja Arduinon kirjastojen yhteensopivuutta ATmega328P-prosessorin kanssa ryhdittiin jo epäilemään, vaikka Arduinon korteissa käytetään samaa prosessoria.

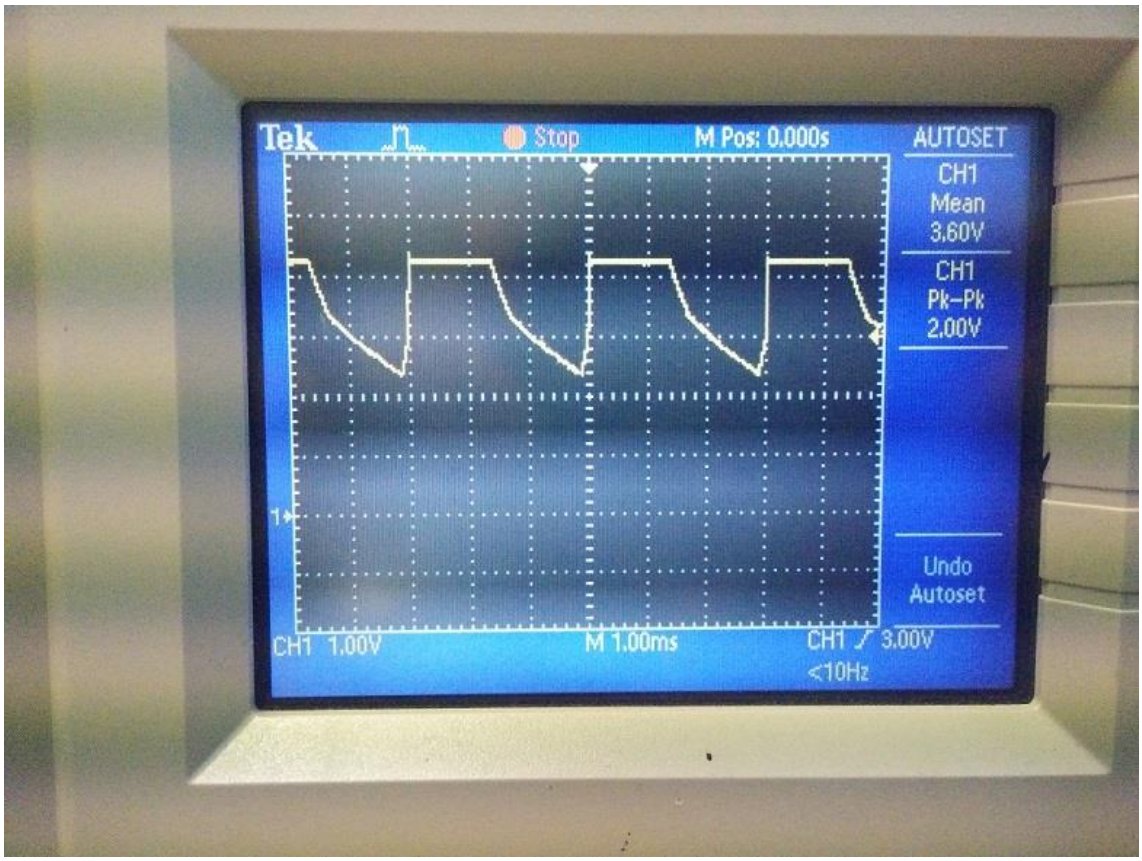
Prosessoriin asetetut sulakkeet oli mahdollista nähdä Atmel Studio 6.2:sta käyttämällä erillistä Olimex AVR-ISP500 -ohjelmointilaitetta. Sulakkeet oli asetettu samalla tavoin kuin Arduino Nanossa, joten prosessorin olisi pitänyt toimia täysin samalla tavalla kuin ensimmäisessäkin prototyypissä. Uudelleenkäynnistyminen viittasi Watchdog Timer -ohjelmallisuuden käyttöön: WDT käynnistää prosessorin uudelleen tai aiheuttaa keskeytyksen, jos se ei ole saanut prosessorilta vastausta tietyin aikavälein [4, s. 51]. Ohjelmallisuus ei ollut kuitenkaan päällä, eikä WDT:n manuaalinen poiskytkentäkään vaikuttanut millään tavalla prosessorin ulosantiin.

Vikaa lähdettiin etsimään poissulkemismenetelmällä toteuttamalla UART-pohjainen tiedonsiirto, jolloin Arduinon kirjastot voitaisiin sulkea pois laskuista. Sulakkeita tutkittaessa huomattiin kuitenkin muutama seikka, jotka paljastivat laitteistossa piilevän vian. Kun prosessori kytkettiin Atmel Studioon ohjelmointilaitteen kautta, näytti Studio samalla jännitteen, joka prosessoriin oli kullakin hetkellä syötettynä. Käytön aikana prosessorin käyttöjännite laski alle 2,5 V:iin, mikä herätti epäillykset siitä, onko prosessori rikki.

ATmega328P-prosessoreihin sisältyvä sulake BODLEVEL pitää kirjaa prosessorin jännitetasosta, ja sen tarkoituksena on täten estää datan vääristyminen ja EEPROM-muistin korruptoituminen [4, s. 21]. BODLEVEL käyttäytyykin niin, että prosessorille asetetaan jokin jännitteen raja-arvo, jonka yläpuolella prosessori saa toimia. Jos raja-arvo alitetaan, BODLEVEL sammuttaa prosessorin ja käynnistää sen uudelleen vasta, kun käyttöjännite nousee taas yli raja-arvon. [8.]

BODLEVEL-sulakkeen raja-arvo on säädetty 2,7 V:iin Arduino Nanon sulakkeista, joten voitiin olettaa, että jännitetason alitus aiheutti uudelleenkäynnistymisen. Olettamus osoittautui oikeaksi, sillä kun BODLEVEL-sulakkeen raja-arvo säädettiin 1,8 V:iin, prosessori alkoi taas toimia. Ensimmäiseen prototyyppiin tehty ohjelmisto toimi täysin samalla tavalla kuin Adafruit Pro Trinket -kortin kanssa, ja piiri ei enää käynnistynyt uudelleen itsestään.

Asiaa ryhdyttiin tutkimaan, ja selvisi, että piirilevyssä käytetty regulaattori ei antanutkaan tasaista 3,3 V:n jännitettä pelkän USB-johdon varassa, vaan jännite laski noin kahden millisekunnin välein huomattavasti, mikä voidaan nähdä kuvasta 13.



Kuva 13. Toisen prototyypin regulaattorin ulostulojännite.

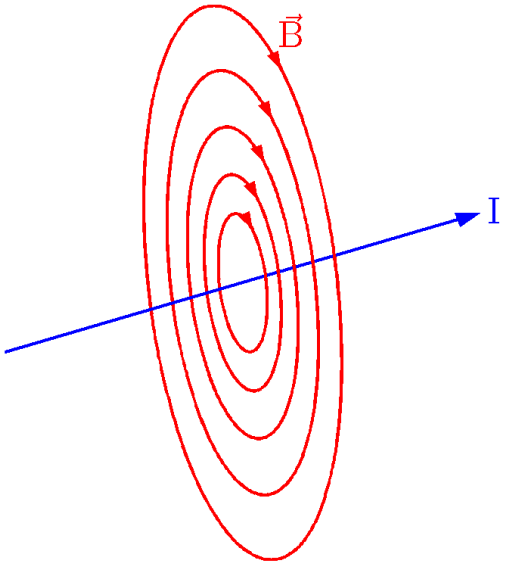
Asiasta ilmoitettiin asiakkaalle, minkä jälkeen selvisi, että piirilevy tarvitsi 3,7 V:n akun toimiakseen stabiilisti kyseisen regulaattorin kanssa. Ongelma osoittautuikin olevan itse piirilevyssä eikä niinkään ohjelmakoodissa, mikä varmistettiin toisen prototyypin kanssa. Kun prototyyppiin kytkettiin 3,7 V:n akku, ongelmat hävisivät, joten lopullisen tuotteen kanssa samaa ongelmaa ei ole.

4.1.2 Värinämoottorit

Tasapainokepin ”lopullista” prototyyppiä päästiin testaamaan projektin loppupuolella. Keppi koostui noin kaksi metriä pitkästä alumiinirungosta, jonka sisällä kulki raiteet. Raiteisiin istutettiin projektin aikana työstetty piirilevy, minkä lisäksi piirilevyn päihin oli lisätty alumiinipalat, jotta rungon sisuskalut ylettyisivät rungon päihin. Samalla kepin molempiin päihin, alumiinipalojen yhteyteen, oli lisätty värinämoottorit.

Ensimmäinen värinämoottoreiden aiheuttama ongelma oli se, että moottoreiden aiheuttama värinä riitti sammuttamaan kepin. Tasapainokepissä on käytössä virtapainikkeena kapasitiivinen sensori, joka kerää häiriöitä itseensä. Moottoreiden värinä kulki runkoa pitkin sensorille ja aiheutti kapasitiivisen yhteyden syntymisen sillä seurauksella, että moottorit sammuttivat jatkuvasti kepin.

Huomion arvoista oli myös se, että moottorit kepin päissä oli yhdistetty piirilevyyn käyttämällä suoria kaapeleita. Kun sähköjohtimissa kulkee sähkövirtaa, ne muodostavat ympärilleen sekä sähkökentän että magneettikentän, jonka kuvasta 14 voi nähdä. Sähkökentän vaikutukset saadaan poistettua eristämällä johtimet, ja siinä suhteessa johtava runko onkin käypä ratkaisu [43, s. 65–66].



Kuva 14. Suora sähköjohdin aiheuttaa ympärilleen magneettikentän [44].

Magneettikenttä tuottaa kuitenkin omat ongelmansa. Magneettikenttä aiheuttaa häiriötä elektronisissa komponenteissa, eikä sen poistamiseksi riitä johtimien eristäminen [43, s. 65–66]. Samalla avoin magneettikenttä toimii tehokkaasti antennina [43, s. 74]. Tämä oli mahdollista huomata esimerkiksi siitä, että keppi käynnistyi satunnaisesti itsestään.

Magneettikenttä tuleekin kumota käyttämällä esimerkiksi kierrettyä parikaapelia, jossa kaapelit kierretään samalta matkalta samanlaisella kierrolla. Toinen johtimista toimii virrankuljetuksessa menosuuntaan ja toinen paluukaapelina. Näin magneettikentät kumoutuvat parikaapelin sisällä eikä magneettikenttä vuoda ulkopuolelle aiheuttamaan häiriötä. [43, s. 78–79.]

Värinämoottorit aiheuttivat myös ongelmia kiihtyvyysanturin kanssa. Moottoreita ei ollut eristetty piirilevystä, vaan niiden värinä aiheutti koko piirilevyn värisemisen. Tämä oli nähtävissä erityisesti nollattaessa keppiä, kun se oli tarkoitus kääntää 90^{o}:n kulmaan. Moottorit vaikuttivat kiihtyvyysanturin antamaan dataan, ja kiihtyvyysanturi ei osassa tapauksista päässyt ikinä haluttuun kulmaan eikä keppiä ollut mahdollista nollata.

4.1.3 Ulkoinen oskillaattori

Projektin puolivälissä laitteistosta vastaava yritys esitti, että keppi laitettaisiin unitilaan (sleep), kun se käännetään 90^{o}:n kulmaan. Tarkoituksena oli kepin ”sammuttaminen” reset-funktioon mentäessä, jolloin keppi voitaisiin jättää pystyyn ja näin akku ei juurikaan kuluisi. Pelkän unitilan toteutus ei kuitenkaan riitä akkukäyttöiseen järjestelmään, sillä se ei täydellisesti katkaise virtaa laitteesta.

Unitilaa ryhdyttiin kuitenkin toteuttamaan asiakkaan toivomuksesta, ja se saatiin toimimaan hyvällä menestyksellä, kunnes prototyyppi lopetti täydellisesti toimintansa. Vaikutti siltä, että prototyyppi olisi mennyt ikuiseen unitilaan, mutta tarkempien tutkimusten tuloksena selvisi, että unitilaan asettaminen aktivoi Watchdog Timer -ohjelmallisuuden eikä unitilaan jääminen ollut siten mahdollista. Oskilloskoopin avulla oli kuitenkin nähtävissä, että prosessori oli täysin eloton.

Lopulta prosessorin rikkoutumisen syy kuitenkin selvisi. Piirilevyyn oli asetettu 16 MHz:n oskillaattori, jota oli käytetty tähän asti. ATmega328P-prosessorin dokumentaatiosta voi nähdä, että prosessorissa käytetyllä 3,3 V:n jännitteellä on turvallista käyttää kellotaajuutta, jonka rajataajuus on noin 13,3 MHz. Tämä on nähtävissä myös kuvasta 2. Kyseessä olikin melkoinen ylikellotus, sillä turvalliset raja-arvot ylitettiin lähes 20 %:lla. Tämän jälkeen siirryttiin käyttämään prosessorin sisäistä 8 MHz:n taajuutta, minkä jälkeen ongelmia ei enää ilmennyt.

4.1.4 Akun lataus

Yksi projektin aikana ilmenneistä ongelmista liittyi akkulaturiin. Asiakkaalla testikäytössä ollut tasapainokeppi oli odottamattomasti kuollut, ja se tuotiin tutkittavaksi. Keppiä tutkittaessa huomattiin, että siinä käytetty akku oli tyhjentynyt, mikä selitti kepin toimimattomuuden.

Akkua yritettiin ladata, mutta akun yhteydessä ei ollut minkäänlaista suojauspiiriä, joten se oli päässyt purkautumaan lähelle yhtä volttia eikä varaus enää pysynyt akussa, vaikka se ladattiin kolmeen volttiin. Suojauspiirin puute saattaa aiheuttaa ongelmia myös ylilatauksen kanssa, sillä on tavallista, että laitteita jätetään latureihin kiinni. Ylilataus aiheuttaa tietynlaisten akkujen kanssa pahimmassa tapauksessa räjähdysvaaran [45].

Samalla prototyypissä oli toinen ongelma, sillä piirin latausvirta on liian suuri 3,7 V:n akuille. Täten piiriin tuli lisätä esimerkiksi 4,7 kΩ:n suuruinen etuvastus, joka estää latausvirran kasvamisen liian suureksi eikä näin tuhoa akkua eikä muuta piiriä.

4.1.5 Yksinäinen kiihtyvyysanturi

Kiihtyvyysanturin käyttö kallistuskulman mittaamiseksi on toimiva ratkaisu, mutta ilman gyroskooppia kiihtyvyysanturin antamaa dataa ei aina voida pitää luotettavana. Gyroskoopin tarkoituksena on kertoa sen orientaatio eli missä asennossa gyroskooppi oikeasti on [46].

Kiihtyvyysanturi mittaa nimensä mukaisesti kiihtyvyyttä: siten kiihtyvyysanturin tuottamaan dataan voidaan vaikuttaa esimerkiksi nopeilla liikkeillä ja tärähdyksillä [3]. Testien aikana todettiin, että esimerkiksi peliin liittyvät hyppyliikkeet tuottivat ongelmia kiihtyvyysanturin datan luotettavuuden kannalta: vaikka keppi pysyisikin tasapainoasemassa, saattaa anturiin muodostua voimia, jotka väittävät sen kallistuneen voimakkaasti.

4.1.6 Alumiinirunko

Ensimmäisenä havaittu ongelma oli se, että alumiini materiaalina johtaa todella hyvin sähköä [47]. Kepin sisältämää laitteistoa ei ollut maadoitettu alumiinirungon kanssa samaan maahan, joten elektroniikan ja rungon välille saattaa muodostua potentiaalieroja. Rungon varautuessa esimerkiksi hankaussähkön vaikutuksesta potentiaaliero saattaa häiritä kepin sisältämän elektroniikan toimintaa. [43, s. 89.] Potentiaaliero on ongelmallinen myös siinä mielessä, että sähkövirta pyrkii purkautumaan ulos helpointa kautta, yleiseensä terävästä kulmasta [43, s. 304].

Ohjelmointiin ja sarjaliikenteeseen tarkoitetut nastat oli jätetty piirilevyn pohjaan avoimiksi. Nastat ovat teräviä, joten runkoon varautunut sähkövirta saattaa purkautua nastojen kautta suoraan piirilevyyn [43, s. 304]. Staattinen sähköpurkaus saattaa tuhota pienjännite-elektroniikkaa, joten korjaamattomana laitteiden tuhoutuminen käytössä on mahdollista [48].

Piirilevyä ei ollut eristetty millään tavalla rungosta lukuun ottamatta osia, jotka olivat suorassa kontaktissa alumiinirungon tai muiden sähköä johtavien osien kanssa. Siten piirilevy saattaa muodostaa kondensaattorin alumiinirungon kanssa ja kerätä häiriöt sitä kautta [43, s. 48–49].

4.2 Luovutettu tuote

Asiakkaalle luovutettiin määritelmien mukainen tuote, johon oli lisätty seuraavat kokonaisuudet:

- kiihtyvyysdatan lukeminen ja suodatus
- luettujen kiihtyvyyksien muuttaminen kallistuskulmaksi
- pistelaskuri ja keppiin yhdistettyjen näyttöjen toiminta
- värinämoottoreiden ohjaaminen
- Bluetooth-moduulin käyttö tiedonsiirtoon.

Kiihtyvyysanturilta saatu data siirretään ensin prosessorille, missä se suodatetaan käyttämällä ohjelmallista alipäästösuodatusta. Suodatukseen käytettiin kaavaa 7:

$$a_n = a_n \times \beta + (a_{n-1} \times (1 - \beta)) \quad (7)$$

Kaavassa a_n on mitattu kiihtyvyys, a_{n-1} edellinen mitattu (ja suodatettu) kiihtyvyys ja β kerroin, jolla suodatuksen voimakkuutta on mahdollista ohjata. Mitä suurempi arvo kertoimelle annetaan, sitä voimakkaampaa suodatusta käytetään. Toisin sanoen suurilla kertoimilla muutokset tapahtuvat hitaasti eikä nopeisiin muutoksiin reagoida juuri ollenkaan. [49.] Jokainen akseli suodatetaan luovutetussa tuotteessa kertoimella 0,25, eli nopeimmat muutokset, esimerkiksi käsien tärinä ja muut värinät, suodatetaan pois, mutta tasapainokeppi reagoi kuitenkin nopeasti käyttäjän antamaan syötteeseen.

Kiihtyvyyksien suodatuksen jälkeen lasketaan tasapainokepin kallistuskulma. Kulman laskentaan käytetään luvussa 3.1.1 esiteltyä kaavaa. Vasemmalle kallistuessaan kallistuskulma saa negatiivisia arvoja väliltä −90...0, kun taas oikealle kallistuessaan arvot rajoittuvat välille 0...90. Tasapainokepin kierto ei vaikuta kallistuskulman laskentaan, jolloin tasapainokeppiä voidaan käyttää ilman, että näytöt osoittavat ylöspäin.

Kallistuskulman määrittämisen jälkeen kasvatetaan tarvittaessa virhepistelaskuria. Oletusasetuksia käytettäessä laskurin arvo kasvaa, kun kallistuskulman itseisarvo on 9:n ja 81:n välillä. Virhepistelaskuri aloittaa asiakkaan toiveesta laskentansa luvusta 999 ja laskee virheet kohti nollaa. Peliä pelattaessa virhepistelaskuri kasvaa väistämättä, vaikka suoritus olisi täydellinen, sillä kepissä toteutettu nollaus on kepin kääntäminen 90°:n kulmaan. Tämä otetaan huomioon pelin puolelta kompensoimalla pistemenetystä lisäämällä pelaajan pisteisiin ennalta määritelty pistemäärä.

Virhepistelaskurin arvo kirjoitetaan näyttöihin kolmen kierrossa. Ensimmäiseen näyttöön kirjoitetaan sadat, jotka saadaan jakamalla pistelaskurin arvo sadalla. Toiseen näyttöön kirjoitetaan kymmenet, jotka saadaan ottamalla luvusta ensin sadan jakojäännös, minkä jälkeen saatu tulos jaetaan vielä kymmenellä. Kolmanteen näyttöön kirjoitetaan ykköset, jotka saadaan ottamalla luvusta ensin sadan jakojäännös, minkä jälkeen luvusta otetaan vielä kymmenen jakojäännös. Lopulliseen tuotteeseen tulevien näyttöjen päälle tulee kansi, jonka lävitse numerot näkyvät selkeästi ja kirkkaina.

Värinämoottoreiden ohjaus toteutetaan nostamalla pystyyn sen moottorin lippu, jonka puolelle tasapainokeppi on kallistettu. Tämän jälkeen moottoreita pulssitetaan 100 millisekunnin päälle/pois-kierrolla. Värinämoottorit tuntuvat hyvin alumiinirungon

lävitse, joskin ilman kunnollista puristusta runko rämisee edelleen ja vaikuttaa kiihtyvyysanturin tuottamaan dataan.

Bluetooth-moduuli mainostaa itseään kepin käynnistyksen jälkeen kaksi minuuttia, minkä jälkeen moduuli lopettaa mainostamisensa. Bluetooth-yhteyden kautta on mahdollista sekä vaikuttaa tasapainokepin toimintaan muuttamalla keppiin määriteltyjä arvoja että lukea tasapainokepiltä tulevia arvoja, jotka lähetetään jokaisen nollauksen yhteydessä. Bluetooth-moduulin kantama on testattu alumiinirungon sisältä ja sen todettiin olevan ainakin 15 metriä, joka on täysin riittävä käyttötarkoitukseen nähden.

Virrankulutukseksi mitattiin 190 mA ohjelmalla, joka pulssittaa moottoreita ja lähettää tietonsa Bluetooth-moduulin kautta. Nykyiset akut ovat kapasiteetiltaan yleensä vähintään 1800 mAh:n suuruisia, joten yhdellä latauksella tasapainokeppi kestää käytössä noin 9,5 tuntia. Lisäksi suurin virtaa kuluttava yksittäinen asia on värinämoottorit, jotka ovat käytössä vain silloin, kun keppi on kallistettuna yli määritellyn raja-arvon. Siten tasapainokepin käyttöaika yhdellä latauksella tulee olemaan käytännössä vielä mainittua suurempi.

5 Yhteenveto

Insinöörityön tarkoituksena oli luoda asiakkaalle tasapainokeppi, jota voitiin käyttää sekä erillisenä projektina toteutetun pelin pelaamiseen että itsenäisesti kuntoilun apuna. Prototyypin oli määrä toimia yhtä lailla molemmissa käyttötarkoituksissa, joten kepin toteutuksessa jouduttiin tekemään kompromisseja.

Asiakkaalle luovutettu tuote mittasi kiihtyvyysanturilla x-, y- ja z-suuntaiset kiihtyvyydet, joiden perusteella laskettiin tasapainokepin kallistuskulma. Kallistuskulman perusteella ohjattiin keppiin asennettuja värinämoottoreita ja kasvatettiin peliin rakennettua virhepistelaskuria. Laskuri laski pisteensä alaspäin luvusta 999 päätyen aina lukuun 0. Laskurin arvo tulostettiin kolmelle erilliselle näytölle, jotka toimivat yhdessä multipleksausta käyttäen. Kun pelattu kenttä oli suoritettu loppuun tai peliohjain haluttiin nollata, se käännettiin 90^{o}:n kulmaan. Tällöin virhepisteiden määrä ja suurin kallistuskulma, jossa tasapainokeppi oli liikkeiden aikana käynyt, lähetettiin Bluetooth-moduulin kautta pelille, jos peliohjain oli siihen yhdistettynä. Tämän jälkeen tasapainokeppi jatkoi toimintaansa normaalisti.

Lopullinen, toinen prototyyppi saatiin valmiiksi keväällä 2016, ja se sisälsi kaikki asiakkaan toivomat ominaisuudet. Lisäksi tasapainokeppiin ohjelmoitiin ylimääräisenä ominaisuutena asetusten käyttökertakohtainen muutos Bluetooth-moduulin kautta. Täten esimerkiksi vaikeusasteen säätö oli mahdollista pelistä käsin, sillä kallistuskulmaa, jolla peli aloittaa virhepisteiden laskemisen, ja laskuriin liittyvää kerrointa oli mahdollista muuttaa ilman, että peliohjaimen lähdekoodia käydään muuttamassa.

Prototyypin toiminta testattiin sekä pelin kanssa että ilman peliä, minkä lisäksi asiakas testasi tasapainokepin toiminnan luovutustilaisuuden yhteydessä. Vaikka toteutuksessa jouduttiin tyytymään muutamaan kompromissiin, olivat molemmat osapuolet tyytyväisiä aikaansaatuun tuotteeseen.

Peliohjaimeen on suunnitteilla lisää ominaisuuksia, joista tärkeimpänä voidaan pitää gyroskoopin lisäämistä kiihtyvyysanturin tueksi. Tällöin tasapainokepin orientaatio saadaan selvitettyä jokaisella hetkellä tarkemmin, mikä auttaa muun muassa tunnistamaan tärähdyksiä. Tärähdysten tunnistaminen on oleellista keppiin toteutetun nollauksen kannalta, ja gyroskooppi lisää tulosten tarkkuutta. Samalla päästään eroon ongelmas-

ta, jossa tärähdysten aiheuttamat muutokset kiihtyvyydessä vääristävät kiihtyvyysvektoreista laskettua kallistuskulmaa.

Lisäominaisuuksien lisääminen saattaa kuitenkin aiheuttaa ongelmia jatkossa, sillä ATmega328P-prosessorista alkaa loppua laskentateho. Prosessori oli riittävä tässä projektissa, mutta koska se on 8-bittinen, sillä ei ole mahdollista suorittaa suurempaa laskentaa. Siten prosessori on syytä vaihtaa tehokkaampaan, jos peliohjaimeen ollaan tekemässä suuria lisäyksiä.

Lähteet

1 MMA8451Q, 3-axis, 14-bit/8-bit Digital Accelerometer. 2015. Datalehti. Freescale Semiconductor.

2 Accelerometer Basics. 2013. Verkkodokumentti. TONI_K, SparkFun. https://learn.sparkfun.com/tutorials/accelerometer-basics. 28.3.2013. Luettu 13.6.2016.

3 Goodrich, Ryan. 2013. Accelerometers: What They Are & How They Work. Verkkodokumentti. http://www.livescience.com/40102-accelerometers.html. 1.10.2013. Luettu 13.6.2016.

4 ATmega48A/PA/88A/PA/168A/PA/328/P. 2015. Datalehti. Atmel.

5 ATmega328P. Verkkodokumentti. Atmel. http://www.atmel.com/devices/atmega328p.aspx. Luettu 9.2.2016.

6 Memory. Verkkodokumentti. Arduino. https://www.arduino.cc/en/Tutorial/Memory. Luettu 14.6.2016.

7 Benchoff, Brian. 2012. AVR FUSE BITS EXPLAINED. Verkkodokumentti. http://hackaday.com/2012/08/30/avr-fuse-bits-explained/. 30.8.2012. Luettu 3.3.2016.

8 AVR Fuses Explained. 2013. Verkkodokumentti. CryoArchive. http://cryoarchive.net/tutorials/avr-tutorials/avr-fuses-explained/. 13.10.2013. Luettu 3.3.2016.

9 Townsend, Kevin, Cufí, Carles, Akiba & Davidson, Robert. 2014. Getting Started With Bluetooth Low Energy. Verkkodokumentti. https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/ch01.html. 30.4.2014. Luettu 12.6.2016.

10 BL600 SERIES. Verkkodokumentti. LairdTech. http://www.lairdtech.com/products/BL600-Series. Luettu 12.6.2016.

11 Klein, Matt. 2015. What's the Difference Between 2.4 and 5-Ghz Wi-Fi? (and Which Should You Use). Verkkodokumentti. http://www.howtogeek.com/222249/whats-the-difference-between-2.4-ghz-and-5-ghz-wi-fi-and-which-should-you-use/. 28.7.2015. Luettu 13.6.2016.

12 Electric Power. 2013. Verkkodokumentti. JIMB0, SparkFun. https://learn.sparkfun.com/tutorials/electric-power. 26.6.2013. Luettu 12.6.2016.

13 Why iBeacons has developers excited about Bluetooth Low Energy. 2013. Verkkodokumentti. Mubaloo, DeveloperTech. http://www.developer-tech.com/news/2013/oct/07/why-ibeacons-has-developers-excited-about-bluetooth-low-energy/. 7.10.2013. Luettu 12.6.2016.

14 Poole, Ian. Link Budget. Verkkodokumentti. http://www.radio-electronics.com/info/propagation/path-loss/link-budget-calculation-formula-equation.php. Luettu 13.6.2016.

15 BL600 Series – Bluetooth v4.0 Single-Mode Modules with smartBASIC. Tuote-esite. LairdTech.

16 BL600 Development Kit – Using Virtual Serial Port Service (vSP) with smart-BASIC. 2013. Datalehti. LairdTech.

17 The AVR TWI (I^2C) Interface. 2003. Verkkodokumentti. AVRbeginners. http://www.avrbeginners.net/architecture/twi/twi.html. 27.8.2003. Luettu 17.3.2016.

18 Adafruit Pro Trinket – 3V 12MHz. Verkkodokumentti. Adafruit. https://www.adafruit.com/products/2010. Luettu 16.2.2016.

19 I think I fried my FT232R USB UART. 2014. Verkkodokumentti. PhillyNJ, AV-RFREAKS. http://www.avrfreaks.net/forum/i-think-i-fried-my-ft232r-usb-uart. 17.10.2014. Luettu 1.9.2016.

20 Adafruit MMA8451 Accelerometer Breakout – Pinouts. 2014. Verkkodokumentti. lady ada, Adafruit. https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/pinouts. Päivitetty 4.5.2015. Luettu 16.2.2016.

21 Fundamentals of Voltage Regulators. Verkkodokumentti. Analog Devices. http://www.analog.com/en/products/landing-pages/001/fundamentals-of-voltage-regulators.html. Luettu 16.2.2016.

22 Libraries. Verkkodokumentti. Arduino. https://www.arduino.cc/en/Reference/Libraries. Luettu 16.2.2016.

23 Using Windows Phone 7 Technologies: Understanding Orientation and Move-ment. 2011. Verkkodokumentti. Programming 4 Us. http://programming4.us/mobile/2220.aspx. 24.1.2011. Luettu 16.2.2016.

24 AVR ISP Header Pinouts. 2012. Verkkodokumentti. Batsocks. http://www.batsocks.co.uk/readme/isp_headers.htm. 15.6.2012. Luettu 1.3.2016.

25 Serial Peripheral Interface (SPI). 2013. Verkkodokumentti. MIKEG RUSIN, SparkFun. https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi. 14.1.2013. Luettu 1.3.2016.

26 AVR910: In-System Programming. 2008. Datalehti. Atmel.

27 Installing an Arduino Bootloader. 2013. Verkkodokumentti. M-SHORT, Spark-Fun. https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader. 4.12.2013. Luettu 1.3.2016.

28 Currey, Martyn. 2014. Arduino Nano as an ISP Programmer. Verkkodokumentti. http://www.martyncurrey.com/arduino-nano-as-an-isp-programmer/. Päivitetty 12.5.2015. Luettu 1.3.2016.

29 AVR-ISP500. Verkkodokumentti. Olimex. https://www.olimex.com/Products/AVR/Programmers/AVR-ISP500/. Luettu 1.3.2016.

30 Currey, Martin. 2014. Arduino / ATmega 328P fuse settings. Verkkodokumentti. http://www.martyncurrey.com/arduino-atmega-328p-fuse-settings/. 16.8.2014. Luettu 3.3.2016.

31 Choudhary, Himanshu. How to program a microcontroller | How to burn a microcontroller. Verkkodokumentti. http://www.engineersgarage.com/tutorials/how-to-program-a-microcontroller. Luettu 3.3.2016.

32 Using the I2C Bus. Verkkodokumentti. Robot-Electronics. http://www.robot-electronics.co.uk/i2c-tutorial. Luettu 14.3.2016.

33 Rouse, Margaret. 2005. Polling. Verkkodokumentti. http://whatis.techtarget.com/definition/polling. Päivitetty 4/2005. Luettu 16.8.2016.

34 7-segment Display. 2013. Verkkodokumentti. ElectronicsTutorials. http://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html. 21.10.2013. Luettu 13.6.2016.

35 Harris, Tom & Fenlon, Wesley. How Light Emitting Diodes Work. Verkkodokumentti. http://electronics.howstuffworks.com/led1.htm. Luettu 16.8.2016.

36 DCN – Multiplexing. Verkkodokumentti. Tutorialspoint. http://www.tutorialspoint.com/data_communication_computer_network/physical_layer_multiplexing.htm. Luettu 13.6.2016.

37 Arduino Timer Interrupts. 2012. Verkkodokumentti. amandaghassaei, Instructables. http://www.instructables.com/id/Arduino-Timer-Interrupts/?ALLSTEPS. 9.7.2012. Luettu 17.3.2016.

38 Koopman, Phil. 2013. Why Short Interrupt Service Routines Matter. Verkkodokumentti. http://betterembsw.blogspot.fi/2013/04/why-short-interrupt-service-routines.html. 25.4.2013. Luettu 17.3.2016.

39 Tuupola, Mika. 2011. Simple Serial Communications With AVR Libc. Verkkodokumentti. http://www.appelsiini.net/2011/simple-usart-with-avr-libc. Päivitetty 7.3.2016. Luettu 18.2.2016.

40 The USART of the AVR. 2013. Verkkodokumentti. Yash, maxEmbedded. http://maxembedded.com/2013/09/the-usart-of-the-avr/. 30.9.2013. Luettu 18.2.2016.

41 Townsend, Kevin. 2014. Introduction to Bluetooth Low Energy – GATT. Verkkodokumentti. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. Päivitetty 4.5.2015. Luettu 12.6.2016.

42 Leach, Paul, Mealling, Michael & Salz, Rich. 2005. A Universally Unique IDentifier (UUID) URN Namespace. Verkkodokumentti. https://tools.ietf.org/html/rfc4122. 7/2005. Luettu 12.6.2016.

43 Ott, Henry W. 2009. Electromagnetic Compatibility Engineering. New Jersey: John Wiley & Sons.

44 Wolski, Andrzej. 2011. Maxwell's Equations for Magnets. Verkkodokumentti. http://inspirehep.net/record/891330/plots. 3.3.2011. Luettu 13.6.2016.

45 BU-409: Charging Lithium-ion. 2010. Verkkodokumentti. Battery University. http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries. Päivitetty 3.8.2016. Luettu 29.8.2016.

46 Goodrich, Ryan. 2013. Accelerometer vs. Gyroscope: What's the Difference? Verkkodokumentti. http://www.livescience.com/40103-accelerometer-vs-gyroscope.html. 1.10.2013. Luettu 29.8.2016.

47 Properties table of Stainless steel, Metals and other Conductive materials. Verkkodokumentti. TIBTECH innovations. http://www.tibtech.com/conductivity.php. Luettu 29.8.2016.

48 Hoffman, Chris. 2013. How to Protect Your PC's Hardware From Static Electricity When Working On It. Verkkodokumentti. http://www.howtogeek.com/169994/how-to-protect-your-pcs-hardware-from-static-electricity-when-working-on-it/. 12.8.2013. Luettu 29.8.2016.

49 Chatterjee, Kirit. 2014. A simple digital low-pass filter in C. Verkkodokumentti. https://kiritchatterjee.wordpress.com/2014/11/10/a-simple-digital-low-pass-filter-in-c/. 10.11.2014. Luettu 16.2.2016.

ATmega328P-prosessorin sulakkeet (fuset)

SULAKE	NIMI AVATTUNA	TARKOITUS	MAHDOLLISET TILAT
BODLEVEL	Brown-Out Detection	Pitää kirjaa prosessorin käyttöjännitteestä. Jos käyttöjännite alitetaan, sulake sammuttaa prosessorin. Näin voidaan estää vääristyneiden tulosten saaminen ja pitää kirjaa epäsuorasti prosessorin käyttöjännitteestä.	DISABLED: Pois päältä 1V8: 1,8 voltin jänniteraja 2V7: 2,7 voltin jänniteraja 4V3: 4,3 voltin jänniteraja
RSTDISBL	Reset Disable	Muuntaa reset-nastan tavalliseksi I/O nastaksi. Käyttöä ei suositella, jos tuote ei ole jo valmis ja menossa kaupalliseen käyttöön.	On/Off
DWEN	Debug Wire Enable	Muuntaa reset-nastan debug wireksi, jolloin reset-nastaa voidaan käyttää ohjelmiston debuggaamiseen. Ei suositella reset-nastan menettämisen takia, joka johtaa ohjelmoinnin vaikeutumiseen.	On/Off
SPEIN	Serial Program Downloading	Mahdollistaa sarjamuotoisen ohjelmoinnin. Sulake pitää olla päällä, jos ohjelmointi halutaan toteuttaa sarjaportin kautta, mutta se on suositeltaa ottaa pois kaupallisissa tuotteissa.	On/off
WDTON	Watchdog Timer Always On	Laittaa Watchdog Timerin käynnistymään aina käynnistyksen yhteydessä. Samalla mahdollistaa Watchdog Timerin keskeytysfunktion käyttämisen.	On/Off

SULAKE	NIMI AVATTUNA	TARKOITUS	MAHDOLLISET TILAT
EESAVE	EEPROM Save Through Chip Erase	Säilyttää EEPROM-muistin siitä huolimatta, että koko muisti tyhjennetään. Voidaan käyttää säilyttämään haluttu data EEPROM:issa	On/Off
BOOTSZ	Boot Size	Määrittelee muistin koon, joka varataan alkulatausohjelmalle.	256W_3F00: 512 tavua alkaen muistipaikasta 0x3F00 512W_3E00: 1024 tavua aloittaen muistipaikasta 0x3E00 1024W_3C00: 2048 tavua aloittaen muistipaikasta 0x3C00 2048W_3800: 2096 tavua aloittaen muistipaikasta 0x3800
BOOTRST	Boot Reset Vector	Käskee prosessoria suorittamaan alkulatausohjelman käynnistyksen yhteydessä. Sulake pitää aktivoida, jos alkulatausohjelmaa halutaan käyttää.	On/Off
CKDIV8	Divide Clock Internally By 8	Jakaa sisäisen kellosignaalin taajuuden kahdeksanteen osaan.	On/Off
CKOUT	Clock Output on PORTB0	Syöttää prosessorissa käytetyn sisäisen tai ulkoisen kellosignaalin ulos portista PORTB0.	On/Off
SUT_CKSEL	Select Clock Source	Antaa käyttäjän valita kellotaajuuden, jota prosessori käyttää. Samalla määritellään prosessorin käynnistymisaika.	INTRCOSC: Sisäinen RC-oskillaattori EXTLOFXTAL: Ulkoinen matalataajuuksinen kristalli EXTFSXTAL: Ulkoinen Full Swing kristalli EXTXOSC: Ulkoinen matalatehoinen kristalli